DTIC FILE COPY

1

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188
1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS NONE
AD-A218 024	3. DISTRIBUTION/AVAILABILITY OF REPORT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.
	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFIT/CI/CIA- 89-158
6a. NAME OF PERFORMING ORGANIZATION AFIT STUDENT AT Univ of Michigan / 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION AFIT/CIA
6c. ADDRESS (City, State, and ZIP Code)	7b. ADDRESS (City, State, and ZIP Code) Wright-Patterson AFB OH 45433-6583
8a. NAME OF FUNDING/SPONSORING ORGANIZATION / 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification) (UNCLASSIFIED)
THREE-DIMENSIONAL MEDICAL IMAGE ANALYSIS USING LOCAL DYNAMIC ALGORITHM SELECTION ON A MULTIPLE-INSTRUCTION, MULTIPLE-DATA ARCHITECTURE

12. PERSONAL AUTHOR(S)
MARTIN ROBERT STYTZ

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
~~XXXXX~~ THESIS/DISSERTATION	FROM ______ TO ______	1989	476

16. SUPPLEMENTARY NOTATION
APPROVED FOR PUBLIC RELEASE IAW AFR 190-1
ERNEST A. HAYGOOD, 1st Lt, USAF
Executive Officer, Civilian Institution Programs

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

DTIC
S ELECTE FEB 15 1990 D
D

90 02 14 001

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT [✓ UNCLASSIFIED/UNLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL ERNEST A. HAYGOOD, 1st Lt, USAF	22b. TELEPHONE (Include Area Code) (513) 255-2259 / 22c. OFFICE SYMBOL AFIT/CI

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

AFIT/CI "OVERPRINT"

A

ABSTRACT

THREE-DIMENSIONAL MEDICAL IMAGE ANALYSIS USING LOCAL DYNAMIC ALGORITHM SELECTION ON A MULTIPLE-INSTRUCTION, MULTIPLE-DATA ARCHITECTURE

by
Martin Robert Stytz

Co-Chairmen: Gideon Frieder, Bernard A. Galler

The dissertation outlines development of a medical imaging machine which renders 3D images from voxel data within a MIMD multiprocessor architecture at interactive rates. Interactive performance is achieved using local dynamic selection of the optimum adaptive recursive hidden-surface removal algorithm.

A survey of the medical imaging, graphics, and medical imaging modality literature is provided. A description of Computerized Tomography, Magnetic Resonance Imaging, Positron Emission Tomography, Single Photon Emission Computed Tomography, and Ultrasound imaging modalities is presented. Previous work in 3D volume rendering graphics techniques and data models is introduced. Eleven medical imaging machines are examined with emphasis on characterization of the major innovation(s) and performance of each machine.

A five stage image processing pipeline is described. The front-end of the pipeline performs user interface and volume rendering operations in software upon voxel data. The back-end of the pipeline accomplishes pixel shading in hardware. The recursive hidden-

Keywords: medical & computer applications.
(KR)

THREE-DIMENSIONAL MEDICAL IMAGE ANALYSIS USING LOCAL DYNAMIC ALGORITHM SELECTION ON A MULTIPLE-INSTRUCTION, MULTIPLE-DATA ARCHITECTURE

by

Martin Robert Stytz

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1989

Doctoral Committee:

Professor Gideon Frieder, Co-Chairman
Professor Bernard A. Galler, Co-Chairman
Professor Harold Borkin
Professor Aaron Finerman
Assistant Professor Michael J. Flynn
Associate Professor Janice M. Jenkins

surface removal operation in the front-end of the pipeline is found to be the pipeline bottleneck.

To provide a framework for development of improved recursive hidden-surface removal algorithms, a definition of recursive hidden-surface removal algorithms is formulated. Back-to-front and front-to-back recursive hidden-surface removal algorithms suitable for scene editing are developed from this definition and their performance is investigated. A processing termination capability for back-to-front hidden-surface removal algorithms is developed from the definition, the resulting adaptive back-to-front recursive hidden-surface removal algorithm is described and its performance is examined. A processing termination capability for front-to-back hidden-surface removal algorithms is developed from the definition, the resulting adaptive front-to-back recursive hidden-surface removal algorithm is specified and its performance is investigated.

An algorithm to accomplish individual-processor-based dynamic selection of either the adaptive back-to-front or adaptive front-to-back hidden-surface removal algorithm based on the processing situation faced by each processor is described. Dynamic algorithm selection is shown to reduce the amount of time spent rendering the image by allowing each processor to employ the adaptive hidden-surface remova' algorithm with minimum local image rendering time.

To Gayl.

ACKNOWLEDGMENTS

I thank God for His many blessings to me, among which I include the opportunity to undertake and complete the work required to earn a Ph.D. from the University of Michigan. Without His help this work could not have been completed.

Additional thanks are due my committee members, Professor Gideon Frieder, Professor Bernard Galler, Professor Harold Borkin, Professor Aaron Finerman, Professor Michael Flynn, and Professor Janice Jenkins, as their support, encouragement, and insight improved the quality of the work presented herein. I have been privileged to have such people on my committee. I especially am indebted to Dr. Gideon Frieder and Dr. Bernie Galler for their mentorship, assistance, and enthusiasm as well as their editorial and research criticisms. I could not have had better co-chairs.

I also thank the friends I have among the faculty, staff, and students here at the university for their support and encouragement. Professor Terry Weymouth, Professor Quentin Stout, and Mark Segal deserve special thanks. Dan Kiskis has been a good friend in addition to being very helpful. Two people are especially important and are invaluable friends, Ophir Frieder and Julie Scherer. I am particularly grateful to Ophir for his support and professional advice. Julie is a very special friend, I am indebted to her for her encouragement, assistance, and comradery.

Finally, thanks to my wife, Gayl, for enduring the late nights, excuses, and loneliness of being the spouse of a Ph.D. candidate in order for me to complete my degree.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER I
INTRODUCTION AND OVERVIEW

1.1 Prologue.

A patient with intractable seizure activity is admitted to a major hospital for treatment. As the first step in treatment, a set of 63 Magnetic Resonance Imaging (MRI) image slices is gathered, but no abnormality is revealed in any of the 2D slices. A three-dimensional (3D) model of the images from the MRI study reveal that the gyri of the lower motor and sensory strips are flattened, a condition which was not apparent in the cross-sectional MRI views. A second study, using Positron Emission Tomography (PET), is ordered to portray the metabolic activity of the brain. Using the results of the PET study, a 3D model of the average cortical metabolic activity is assembled, which discloses a volume of hypermetabolic activity. However, because PET has relatively poor image resolution, the location of the abnormality can not be related to surface anatomical landmarks. The 3D MRI model of the patient's brain is combined with the 3D PET model using *post-hoc* image registration techniques to align the two sets of data. This combined model provides the anatomical and metabolic information required for precise location of the abnormality within the the lower part of the motor and sensory strips.

A surgery rehearsal program is employed to display the combined brain model and overlying skin surfaces side-by-side on the screen. Using a mouse controlled cursor, the abnormal area of the brain is outlined on the combined model. This tracing is used by the program to perform a simulated craniotomy, with the resulting images displayed on the skin-surface 3D image. Pictures of the simulated operation are taken to surgery to guide the actual procedure. An intraoperative electroencephalogram demonstrates seizure activity at the site predicted by the medical images. After resectioning of the abnormal area, the patient's seizure activity ceased. (from [Hu89]).

1.2 Introduction.

The preceding case study illustrates the three basic operations performed in 3D medical imaging: data collection, 3D data display, and data analysis. The example also points out the difference between medical imaging and computer graphics. Three-dimensional medical imaging attempts to generate faithful graphical representations of a physical volume, while the field of 3D computer graphics strives to form realistic images from scene descriptions which may, or may not, have a physical counterpart. However, the two fields share a common challenge in their attempt to portray a 3D volume within a two-dimensional (2D) space, and hence perform many of the same graphics operations: coordinate transformation, hidden-surface removal, and shading. Coordinate transformation, which consists of translation, rotation, and scaling, orients the volume to be displayed. Hidden-surface removal insures that only the visible portions of the objects are displayed. Shading provides depth cues and enhances image realism. 3D medical imaging utilizes these techniques to generate credible portrayals of interior volumes of patients for disease diagnosis and surgical planning. This dissertation addresses performing 3D medical imaging operations within an Multiple-Instruction, Multiple-Data (MIMD, see [Fly72]) multiprocessor environment.

The demand for accurate images stems from the data analysis step and concerns us because the prior computations must provide the trustworthy visual information that the physician requires. Data collection is the radiologist's arena, and deals with issues concerning the statistical significance of collected data, patient dosage, medical imaging modality operation, development of new techniques/modalities for gathering data, and 2D image reconstruction. 3D data display issues belong to the computer scientist, as these concern computer graphics, throughput, computer architecture, image processing, database organization, numerical accuracy in computations, and the user interface. However, as attested by the number of radiologists and physicians who also deal with display issues, a knowledge of the physician's requirements as well as the capabilities and limitations of the modalities is essential to addressing the questions posed by the image display process.

The difficulty in meeting the physician's requirement for image accuracy and high image processing speed arises from the operation of the modalities. Since image accuracy is limited because the modalities sample space and reconstruct it mathematically, a display can not hope to achieve the accuracy of just looking into the body. The ability to rapidly form the images is retarded by the large volume of data to be processed, with up to 35 Mbytes produced per patient from a single modality. The challenge posed to the computer scientist, then, lies in the development of techniques for rapid, accurate manipulation of these large quantities of data in order to produce images which are useful to a physician.

Traditionally, 3D image formation speed has been sacrificed for reconstructed image accuracy. The first machines from the Medical Image Processing Group in Philadelphia, Pennsylvania in 1979 employed algorithms for data management and image volume reduction by surface detection to provide their displays, but generating a single series of rotated views of a single X-Ray Computerized Tomography (CT) study was an overnight process ([Art79b], [Art81]). Initial attempts at achieving interactive display rates using the unprocessed 3D data set relied upon special-purpose multiprocessor architectures and hardware-based algorithm implementations. While high quality images are generated rapidly with this technique, the drawback to this approach is the placement of image rendering algorithms in hardware. Because 3D medical imaging, and the related area of computer graphics, are such rapidly changing fields, flexible implementations are required so that a machine's image rendering processes can be altered to take advantage of developments in volume rendering techniques.

The comparatively recent advent of high-power, inexpensive processors with large memory address spaces opens the possibility for using software-based algorithm implementations which operate upon the raw data set within a multiprocessor architecture to provide real-time interactive displays. This approach achieves volume rendering accuracy by foregoing volume reduction operations, rapid display rates by distributing the workload, and implementation flexibility by using software to realize the image processing algorithms. However, real-time 3D medical image generation remains an unrealized goal, and issues concerning distribution of workload, data set representation, and appropriate image rendering algorithms remain.

1.3 Research Goals.

The broad aim of this research is to investigate the capability of general-purpose parallel processing architectures to provide an interactive three-dimensional medical imaging environment. In particular, given the computational cost of the hidden-surface removal process in 3D medical imaging, the primary thrust of this work is the development of general-purpose hidden-surface removal algorithms which accelerate the 3D medical image rendering process without sacrificing image accuracy.

1.4 Operational Environment.

To support the research objectives cited above, a basic 3D medical imaging operational environment was assumed. The broad elements of this environment are sketched in this section, specifics are contained in the pertinent sections of Chapter V.

Because of the compute intensive nature of medical image processing, thc decision to locate the image processing algorithms in hardware or in software has a major impact on the performance of the machine. Many prior research efforts aimed at high image rendering speed, such as the Voxel Processor and Fuch's machine, have elected hardware-based algorithm implementations and foregone the flexibility of a software-based approach, to achieve their high-speed performance goals. Those efforts which are primarily software based, for example Farrell's machine, Robb's machine, and the MIPG machines, reduce the magnitude of the image processing task by surface extraction or by reduction in the resolution of the image. The requirement for accurate, rapidly produced images mitigates against this approach. To meet the requirement for speed and accuracy, I decided to use the unprocessed image data within a mixed hardware/software-based implementation.

The motivation for this approach comes from the twin requirements for image accuracy and rapid rendering speed as well as the desire for a flexible implementation. The use of software rather than hardware provides a flexible, modular implementation of the image rendering algorithms. The use of the raw voxel data avoids the information loss inherent in the surface detection approaches to volume reduction. The visual information loss arises from the difficulty in accomplishing surface location and surface tracking. The mixed hardware/software approach places the compute intensive pixel lighting operations in hardware, while the remaining image rendering operations are performed upon the unprocessed imaging modality data in software. This decision, made nearly two years ago, has been supported by recent commercial efforts which have selected the same approach in order to keep pace with the rapidly changing 3D medical imaging and computer graphics fields. There are two drawbacks to the approach I chose, the slowness inherent in a software implementation and the large volume of data to be processed. To achieve interactive processing speeds requires the distribution of the processing workload within a parallel processing environment.

In order to gain the maximum performance benefit from a parallel-processing architecture, the workload must be evenly, or nearly evenly distributed among the processors of the machine. Two approaches to achieve this goal are available, partitioning the data in object space or in image space. I use both types of partitioning. As the visible portion of image space can change greatly from frame to frame, image space partitioning is performed only on the rendered volume. A common approach divides the output image into a set of small squares which are independently processed. As discussed in Chapter IV, this approach is taken in the Stellar™ machines, Fuch's Pixel Planes machines, the Ardent™ Titan™, and the AT&T Pixel Machines™. I employ a similar technique for image space division in the image processing pipeline I use. I use object space partitioning to achieve a balanced workload among the machine's processors, but this imposes the restriction that the algorithms employed in the processors do not require global information. The partitioning itself can be either static, as in [Rey85], or dynamic, as in [Dip84]. I use a static, equal distribution of data among the pipeline front-end processors in order to achieve equal image complexity, and hence processing load, among the processors in all the stages.

The decision to embed the image processing algorithms in software raises the issue of the type of parallel processing performed within the machine and how the processing is organized. The host parallel processing machine is assumed to be a distributed memory MIMD (Multiple Instruction, Multiple Data [Fly72]) architecture machine which uses message packets for internode communication. A MIMD architecture is used because of the two advantages it offers over Single-Instruction, Multiple-Data (SIMD) architectures in graphics applications because it simplifies the coding of the graphics algorithms and is more efficient in its use of processor cycles. These advantages are obtained because the processors can operate independently rather than being required to operate in lockstep upon a single broadcast instruction, as in SIMD. The use of message packets rather than common memory for communication eliminates the memory contention/synchronization issues which affect the common memory machines while permitting inter-process synchronization .

Processing within the MIMD parallel processing machine is organized into a pipeline, with the front-end of the pipeline responsible for the user interface and visible surface detection operations and the back-end devoted to pixel shading operations. This approach segregates the image processing operations into functional units, which makes testing and debugging easier. In addition, the performance of the pipeline can be incrementally improved by attacking the performance of critical, bottleneck stages rather than addressing the image rendering operation as a whole. This approach also addresses the complaint, (see Duf85] for an example), that the design of image rendering machines as monolithic devices makes them exceedingly difficult to understand. The use of a graphics processing pipeline facilitates the use of modular component design and testing by operating the individual stages of the pipeline independently, with stubs to simulate communication with other pipeline stage modules. The validated modules can then be brought together within the MIMD parallel processing machine to generate images, with the only changes to the code being replacement of the testing stubs with calls to send/receive messages.

1.5 Overview.

This dissertation addresses the two main areas of my work, the background information required to place the research into context and the research contributions themselves. The background, contained in Chapters II - IV, provides a general introduction to the area of medical imaging in its treatment of data acquisition modalities, image rendering techniques, and previous designs for medical imaging machines. Based upon this background, the exposition in Chapters V and VI describes the results of my own work upon a medical imaging machine design and the general algorithms developed to reduce image rendering time.

Chapter II contains a description of five medical imaging modalities: Computerized Tomography, Magnetic Resonance Imaging, Positron Emission Tomography, Single Photon Emission Computed Tomography, and Ultrasound. The discussion in that chapter focuses primarily on the imaging aspects of each modality, as this parameter, by and large, determines the modality's clinical utility. Because of its introductory nature, the discussion contains a minimum of formalism and instead provides an intuitive introduction to the operation, physics, and mathematics which underlie the operation of each modality.

Chapter III introduces the graphics techniques and data models utilized to render 3D medical image volumes in order to provide a background for the discussion in Chapters V and VI. The object and image space coordinate systems are explained, and the octree, cuberille, and voxel data models are defined. The chapter contains descriptions of the major approaches developed for performing the fundamental image processing tasks of geometric transformation, hidden-surface removal, shading, object surface isolation, and anti-aliasing within the context of the voxel data model. The concepts and algorithms used in ray tracing are also presented, as the issues involved in the use of ray tracing within the medical imaging machine I propose are addressed within this dissertation. However, rather than provide a comprehensive review of the uses of ray tracing in computer graphics, the discussion is limited to the issues and techniques employed for hidden-surface removal, anti-aliasing, image compositing, and shading in the medical imaging environment.

To provide a setting for the results presented in Chapters V and VI in terms of prior efforts in the 3D medical imaging field, Chapter IV contains descriptions of eleven research and commercial medical imaging machines. Each machine's description concentrates on the major innovation(s) it expresses as well as sketching the machine's approach to the medical image rendering task. The machine characterization consists of a description of the overall machine architecture and degree of parallelism employed in the machine along with a short explanation of the processing strategy it uses on its assumed medical image data model. Each machine's shading algorithm, anti-aliasing algorithm, and hidden-surface removal algorithm are described. The capability of each machine is assessed in terms of the image resolution it supports and the speed with which it renders 3D medical images. In order to highlight the tradeoffs made in each of the machines, general conclusions about each approach are presented and their performance is compared.

Chapter V is the first of two chapters which present the results of my research into performing medical imaging operations within commercially available multiprocessors. The chapter opens with a description of the operating environment used for the research and the image processing pipeline employed to render images. The remainder of the chapter is devoted to an elapsed image rendering time performance assessment of this pipeline. This analysis has two parts. One component of the evaluation is a comparative analysis of the overall performance of the pipeline when it is embedded in the mesh and hypercube interconnection topologies. The other component is an examination of the performance of a variety of recursive hidden-surface removal algorithms in the critical Sub-Scene Generator stage of the pipeline.

Chapter VI presents three new algorithms which, together, accelerate the performance of the medical image processing pipeline described in Chapter V by approximately 40%. The major innovation in these algorithms comes from two sources. First, the algorithms perform adaptive termination of the hidden-surface removal operation, thereby avoiding exhaustive examination of the space containing the image and the associated computational expense. Second, dynamic selection of the algorithm which offers the fastest image production time for the current set of user inputs is performed independently by each processor.

Chapter VII contains a summary of the dissertation, conclusions, and proposals for future work.

Appendix A contains a glossary of the terms used in this work.

1.6 Summary.

This introduction has provided a brief overview of the motivation, goals, and requirements for medical imaging as well as outlining the presentation of my research which appears in the ensuing pages. The next chapter begins the process of filling in the outline presented here with an introduction to the operation and characteristics of medical imaging modalities.

CHAPTER II
MEDICAL IMAGING MODALITIES

2.1 Introduction.

The requirement for diagnostically relevant image data has spurred the development of many techniques for data acquisition over the last thirty years. The recent development of modalities which support three-dimensional imaging has opened a new arena for disease diagnosis through medical imaging. The addition of this third dimension has relieved the physician from the task of reconstructing the image along different viewpoints, either mentally or with computer aid, and has freed them for the tasks of disease diagnosis and surgical planning. Two broad classes of modalities have developed, those which perform anatomical imaging and the modalities which perform biochemical imaging. Soft and bony tissue imaging is accomplished using Magnetic Resonance Imaging (MRI) and Computerized Tomography (CT). The tomographic imaging of organ biochemical function is accomplished with two different emission computed tomographic (ECT) modalities: Single Photon Emission Computerized Tomography (SPECT) and Positron Emission Tomography (PET). In this, the first of three background chapters, these medical imaging modalities are introduced.

This chapter's introduction focuses primarily on the imaging aspects of each modality, as this parameter, by and large, determines its clinical utility. Because of its introductory nature, the discussion is presented with a minimum of mathematics and physics and instead attempts to provide an intuitive grasp of the operation, physics, and mathematics of each modality. Given the broad nature of the issues in image reconstruction, the tomographic image reconstruction methods are necessarily treated briefly. A few of the many publications which address the mathematics of tomographic image reconstruction are the representative books and articles by [Bat83], [Dea83], [Her80b], [Her85b], [Kak87], [Mac83], and [Nat86].

Typically, papers on these modalities present both engineering advances and application results which accrue from the new technique. Because of the ever increasing number of applications for each of the modalities discussed here, current applications for each modality are not discussed. A few of the many publications which devote themselves to a discussion of applications and advancements in these modalities are: *The British Journal of Radiology*, *Journal of Belge Radiology*, *The Journal of Computer Assisted Tomography*, *The Journal of Nuclear Medicine and Allied Sciences*, *The Journal of Nuclear Medicine*, *The Journal of Ultrasound in Medicine*, *Magnetic Resonance Imaging*, *Nuclear Medicine Annual*, *Radiology*, *Ultrasound in Medicine and Biology* , and *Ultrasound Quarterly*.

This chapter is organized to accomplish two goals. First, to introduce these widely used non-invasive techniques for gathering medical imaging data and, second, to provide an appreciation for the data acquisition process for each modality. Each of the following five sections presents a brief history of the modality, followed by a description of the major components of the machine, an intuitive description of the physics employed by the modality, a tutorial on a operation of a prototypical machine during data acquisition, and a discussion of the factors which affect image quality. Relevant references are provided at the opening of each section.

2.2 Computed Tomography Scanners.

The discovery of X-rays in 1895 by Roentgen opened a new era in medicine as it allowed the visualization of the interior of the body without life-threatening surgery. Over the years, there have been numerous evolutionary improvements in X-ray technology, such as the development of sophisticated and more powerful scanning equipment and improved techniques for analyzing the data. Nevertheless, visualization of soft tissue within the human body could not be accomplished, forcing the use of exploratory surgery or other, offtimes painful, low resolution imaging techniques (such as the pneumoencephalogram) to make a diagnosis. A revolution in medical imaging occurred in the 1970's with the development of the Computed Tomography (CT) scanner. An appreciation for the impact that CT scanning made is evident in the fact that since the first clinical X-ray CT scanner for the head, developed by EMI Ltd., was installed in 1971, only 18 years have passed and now the device is available throughout the US. This impact follows from the CT scans' ability to provide the physician with a capability for nonsurgical, painless examination of internal body structures with unprecedented accuracy and versatility [Rob85]. The success of the CT scanner has been so great that it has promoted the acceptance of other high technology medical imaging techniques, such as Positron Emission Tomography and Magnetic Resonance Imaging. To date, although there have been four generations of CT scanners, differentiated mainly by the fan beam and x-ray detector geometries, there have been no conceptual breakthroughs since the original EMI machine.

The following information on Computed Tomography (CT) is drawn from [Bar81], [Gor75], [Her80b], [Mey87], [Mir84], [Rob82], [Rob85a], [Udu83] and [Wel81]. Further information can be found in these books and articles. In particular, [Mey87], [Gor75], [Her80b], and [Rob85] provide an explanation of the image reconstruction algorithms used within the CT scanner that are not repeated here.

A CT scanner is used to both emit and detect the X-rays used in reconstructing the two-dimensional (2D) image slices of the patient's body. Figure 2.1 contains a diagram of a modern CT scanner.

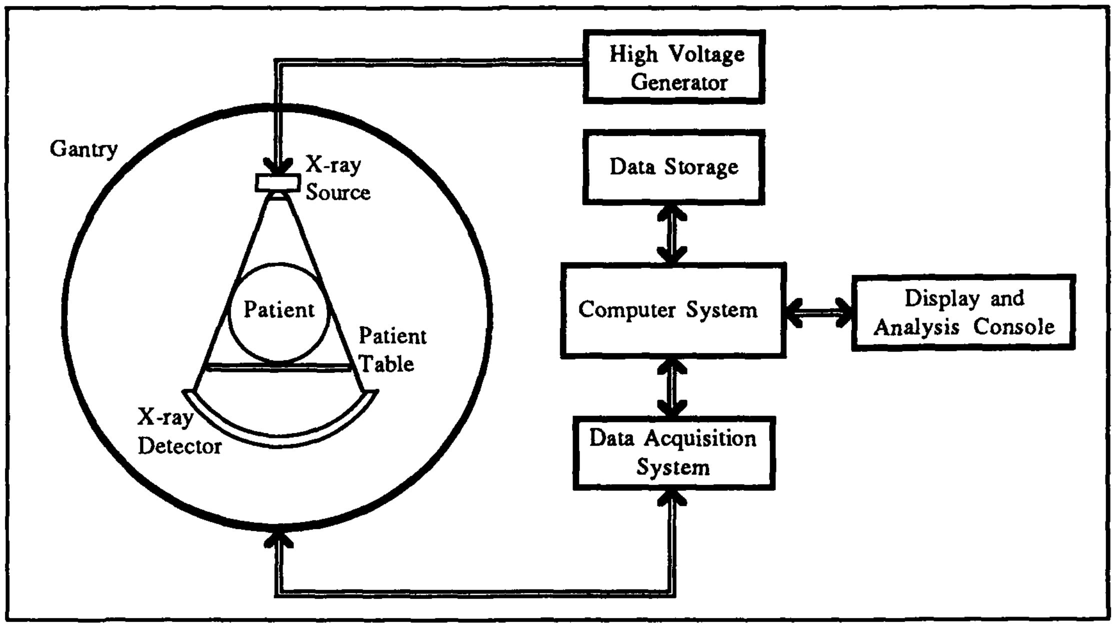


Figure 2.1: X-ray Computed Tomography Scanner Diagram.

The modern CT scanner has seven components: a sliding patient table, a gantry, an X-ray source and collimating assembly, a data acquisition/detector system, a computer system, and a display and analysis console. The patient lies on the table, which is used to precisely position the patient within the gantry as the patient slides into the opening. In a third generation system, the gantry houses the X-ray tube and its collimator assembly on one side and the data acquisition/detector system on the other, all mounted upon a single circular rotating platform. A fourth generation system is virtually identical, except that only the X-ray tube and its collimator assembly rotates and the data acquisition/detector system completely encircles the patient.

The collimator on the x-ray source is used to focus the X-ray beam on the body section being studied The data acquisition/detector system has two parts: a detector unit and an analog-to-digital convertor. The detector unit contains a large number of X-ray detectors arranged on the arc of a circle centered on the X-ray source. The X-ray detectors count the number of X-ray photons which pass through the patient. To reject scattered x-rays, a collimator is placed in front of each detector so that only the x-rays which travel in a nearly straight line are used by the scanner. To date there have been three types of detectors employed, scintillation, gas, and solid-state. Scintillation detectors use sodium iodide (NaI) or bismuth germanate (BGO) to absorb the transmitted x-ray and generate scintillation light which is amplified by a photo-multiplier tube and sent to the analog-to-digital convertor. Gas detectors use xenon or a xenon-krypton gas mixture. The gas produces positive ions when struck by an x-ray. The gas is sealed between metal plates which also determine the aperture size. The number of x-rays which enter each detector is calculated from the current across the plates, as the current is proportional to the number of impinging x-rays. Solid-state detectors operate by emitting an electron when struck by an x-ray; the accumulated charge is proportional to the number of x-rays which strike the detector. The analog-to-digital convertor takes the analog output of the detector unit and converts it to digital form for processing by the computer system. In addition to image reconstruction, the computer system controls patient positioning, rotation of x-ray source and detectors, and other scanning tasks. The display and analysis console is used by the physician to control the image processing functions performed by the computer system and to display the results with up to 4096 gray levels on a 512 x 512 screen in approximately 15 seconds.

The basic methodology employed in CT scanning involves the computation of a cross-sectional distribution of x-ray attenuation in the body by back-projecting the x-ray transmission measurements acquired at many angles around the body. The data for a single transverse slice is acquired by rotating the X-ray source and detector around the patient while gathering image data by emitting X-rays at predetermined locations. Each X-ray detector receives an X-ray signal which has been attenuated by the material on the line from source to detector. The strength of the signal at each detector is given by the integral of the

X-ray attenuation coefficient, μ of the intervening material(s). X-ray attenuation is defined as the the amount of X-ray radiation of a given energy absorbed by an object. The X-ray attenuation coefficient at a single point in the cross-section is indicative of the type of tissue at that point, therefore the attenuation coefficient can be used to form an image that represents the internal structure of a patient. Forming a clinically useful image requires that the attenuation coefficients be known for 10^5 to 10^6 closely spaced (less than 1mm) positions. Since attenuation is only determined after the X-ray has completely traversed the patient, the scanner only has the integral of the X-ray attenuation coefficients along a line to work with, not the required attenuation coefficient for each point in the slice. The X-ray attenuation coefficients for each point in the cross section must, therefore, be calculated from the integrals of the attenuation coefficients. Acceptable image quality is obtained by gathering the integral of the attenuation coefficient for a large number of lines of known position via rotating the source and detector around the patient. To simplify the computational problem, information is collected for one 2D slice at a time and by calculating the attenuation coefficients for small volumes called voxels instead of calculating them for a large number of points in the volume of a single plane. The result of these last two measures is that image quality is reduced, but they permit images to be obtained in a practical length of time (30 minutes or less) with an acceptable amount of radiation exposure.

Image reconstruction can be performed using back-projection, algebraic reconstruction, iterative series expansion, or filtered back-projection. As the filtered back-projection technique is the most commonly employed reconstruction method in CT technology, as well as the other methodologies discussed later, it is described briefly. In 1917 Radon gave an existence proof that an n-dimensional object can be reconstructed from an infinite set of n-1 dimensional projections of the object. This work lay dormant for half a century, until it was retrospectively "rediscovered" after the solution to the image reconstruction problem in the field of X-ray CT was developed by Hounsfield and

Cormack. The reconstruction proceeds through two steps, filtering the raw data and back-projection of the filtered data to form the image. The goal of image reconstruction can be stated as: given a set of line integrals $\{\mathcal{L}\}$, reconstruct the value of the physical parameter measured by the imaging device at each point (r,ϕ)[1]. The function $f(r,\phi)$ represents the value of the physical parameter at each point. For a CT scan, $f(r,\phi)$ is interpreted as the relative linear attenuation at (r,ϕ), whereas in a Positron Emission Tomography (PET) scan, $f(r,\phi)$ is interpreted as concentration of radioisotope. The problem faced in image reconstruction is that $f(r,\phi)$ is unknown and its value at each (r,ϕ) must be estimated from the line integrals. This topic is explored briefly below; comprehensive mathematical treatments of the subject are found in the references mentioned in the introduction to this chapter. The discussion is limited to the parallel beam image reconstruction case and is based on [Her80b], for a discussion of divergent beam geometry and reconstruction issues also see [Her80b].

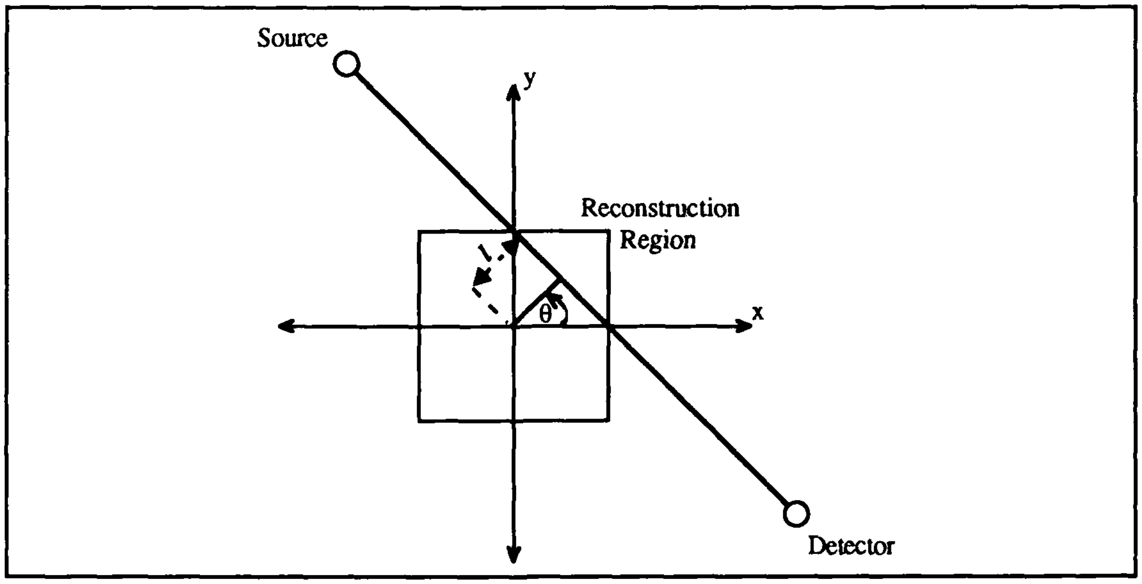


Figure 2.2: Reconstruction Geometry (based upon [Her80b]).

[1]A polar coordinate system is used for notational simplicity.

As image reconstruction is based on line integrals, a notation (from [Her80b]) for specifying the location of the lines is required and is illustrated in Figure 2.2. The location of each line, L, is specified by the angular displacement, θ, of the perpendicular to each line from the x-axis and the distance between the line and the origin, ℓ. Each L is referred to as the line $L_{\theta,\ell}$. Letting $p(\ell,\theta)$ represent the line integral of the function $f(r,\phi)$ along the line $L_{\theta,\ell}$, image reconstruction methods recover $f(r,\phi)$ from $p(\ell,\theta)$ in principle using the inverse Radon transform:

$$f(r,\phi) = \frac{1}{2\pi^2} \int_0^{\pi} \int_{-\infty}^{\infty} \frac{1}{r\cos(\theta - \phi) - \ell}\, p(\ell,\theta)\, d\ell\, d\theta \tag{2.1}$$

Because 2.1 requires an infinite number of samples at ℓ and θ, an approximation to 2.1 is necessary in practice. This approximation is accomplished used a two-step process, filtering[1] followed by back-projection. As p samples are taken at known values of ℓ and θ, a summation expressing $f(r,\phi)$ in terms of the samples can be constructed. If the samples are equally spaced around the circumference of the object, let l_n ($n= 1,\ldots N_p$) and θ_m ($m = 1,\ldots M_p$) represent the locations where the samples were taken. Then $f(r,\phi)$ is approximated by: (from [Her85c])

$$f(r,\phi) \cong \sum_{m=1}^{M_p} w_p(r,\phi,\theta_m) \left[\sum_{n=1}^{N_p} q_p(r,\phi,\ell_n)\, p(\ell_n,\theta_m)\, \Delta\ell \right] \Delta\theta \tag{2.2}$$

The $w_p(r,\phi,\theta_m)$ term is the weighting function which is applied to the results of the filter function $q_p(r,\phi,\ell_n)$. Additional considerations of image accuracy and computational

[1]The entire procedure is termed a convolution.

efficiency result in further refinements to 2.2, but a discussion of these is beyond the scope of this chapter.

A physical interpretation of 2.1 and 2.2 is portrayed in Figure 2.3. In order to provide an estimate of f(r,ϕ), x-rays are passed through an object and the transmitted

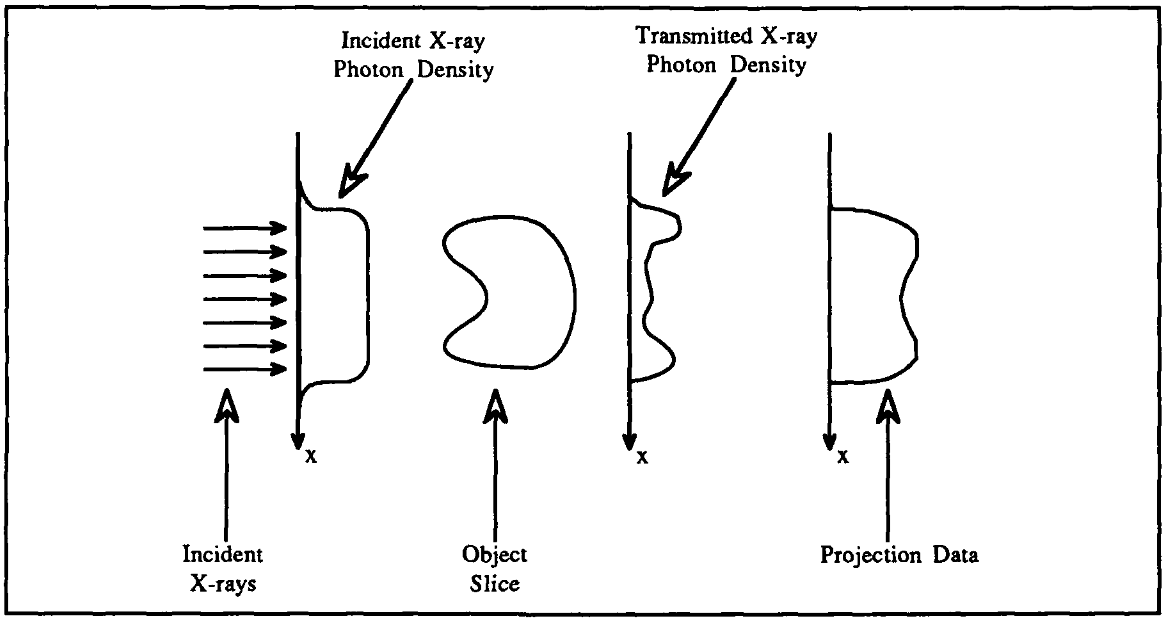

Incident X-ray Photon Density
Transmitted X-ray Photon Density
x
x
x
Incident X-rays
Object Slice
Projection Data

Figure 2.3: Acquisition of Linear Projection Data (based upon [Bar81]).

x-ray photon density measured. The x-ray photon density, in turn, provides the line integrals (projection data) required to estimate f(r,ϕ). In the case of CT, the line integrals correspond to the total attenuation experienced by an x-ray as it traverses the body. Filtering and back-projection are, then, means for estimating the attenuation at each point in the image given the amount of signal lost traversing the object. This is accomplished as follows. Conceptually, the data for each line integral is back-projected (smeared) into a 2D image matrix to form the reconstructed digital planar image. Because the linear projection data does not contain information regarding the 2D location of the source(s) of the data, the projected data is distributed among those elements of the matrix which lie along the line of the projection using the weighting function. Each point of the final digital image now contains the sum of all the back-projected data which was assigned to the point. This forms an unfiltered image. Unfiltered back-projection forms an image with a characteristic star-like appearance and appears blurred. To unblur the image and mitigate against the appearance of the star, the projection data is first filtered using a filter function. The filter function produces a new data value for each data projection from a weighted sum of the data values in the original projection. After the filtering function has been applied, the back-projection operation is performed to form the final digital image.

The amount of attenuation within the voxel is represented by a number, called a Hounsfield Unit (HU), which represents the attenuation within the volume relative to the attenuation of the same volume of water. Hounsfield Unit values range from -1000 to +3000, with a value of zero assigned to the attenuation of water. A HU value of -1000 corresponds to minimal signal attenuation, while a HU of +3000 indicates high attenuation. Within each slice, the voxels are assigned a HU value based on the magnitude of the X-ray attenuation coefficient computed for that volume by the microprocessor in the CT scanner. For voxels falling within slices, the HU is proportional to the average relative linear attenuation within the corresponding volume in the patient.

The standard CT scanner image output is a set of 2D transverse[1] slices of the patient, however the radiologist and/or physician must interpret the image in a 3D manner. With additional computation, a 3D image can be formed from the 2D cross-sectional slices produced by the CT scanner. Since the cross-sectional slices are not contiguous, the inter-slice voxel HU values must be estimated. Three common approaches to this problem are voxel extension, linear interpolation, and trilinear interpolation. The voxel extension scheme is the easiest to implement, the value of voxels falling between slices are assumed to be the value of the closest voxel in the preceding slice. Linear interpolation estimates the values of inter-slice voxels by a weighted average of the values assigned to the two nearest

[1]A transverse slice is taken perpendicular to the patient's long axis.

voxels in the two adjoining slices. Trilinear interpolation computes the inter-slice voxel values as a weighted average of the eight closest voxels from the two adjoining slices. Whichever method is selected, the HU assigned to each inter-slice voxel remains within the -1000 to +3000 HU range.

In 3D space, the location of each voxel is described using a 3D coordinate system. The z-axis is parallel to the CT scanner table, and also to the patient's longest dimension. The y-axis is perpendicular to the top of the CT scanner table. The x-axis is parallel to the opening in the gantry. For image processing purposes, the voxel values reconstructed by the microprocessor and computed from any interpolation are scaled from the 12-bit -1000 to +3000 HU range to an 8-bit, 0 to 256 gray-scale. If a false color image is to be formed, the 12-bit HU range is scaled to an 8-bit value for each of the red, green, and blue color channels. The 3D volume is represented by a 3D array. Each array element represents a single voxel, and the value of the array entry is the corresponding scaled Hounsfield Unit value for the voxel.

The major factors which affect CT scanner image quality are radiation dose, resolution, slice thickness, beam hardening, beam scatter, the image reconstruction algorithm, and imaging artifacts. Increasing the radiation dose to the patient reduces the noise in the image, thereby improving its quality. An increase in dose can also be used to improve resolution, but to keep noise at the same level as in the original image and double the resolution, the dose must increase by a factor of eight. Since the effect of image noise is inversely proportional to the square root of the dosage, the dosage must be increased by a factor of four just to reduce the noise by a factor of two. As the human body is intolerant of X-rays, taking the straightforward approach of increasing the X-ray dose until a high quality image is formed is ruled out. Rather, the effectiveness of the dose has been increased by making better use of the X-ray photons that pass through the patient. The effectiveness of the dose is described by a parameter called the dose efficiency. The dose efficiency indicates how much of the radiation that impinges upon the detector unit is actually captured for image reconstruction. The higher the dose efficiency, the better the quality of the reconstructed image for a given dose. Current CT systems typically have a dose efficiency of between 50% and 70%, thereby providing high quality reconstructions while maintaining the dose to the patient within the safe dosage limit.

The resolution in the CT image is influenced by the X-ray focal spot size, the size of the detector element aperture, the sample spacing, the type of convolution filter, and the size of the reconstruction pixels in the CRT display. The HU range affects the quality of the final image by determining how effectively the CT scanner portrays the actual attenuation within the body. The scanner itself has an attenuation accuracy of $\pm 1/4\%$, or 2.5 Hounsfield Units, that is; the computed attenuation for a given volume is within 1/4% of the actual attenuation for the volume. The challenge, therefore, is to provide a Hounsfield Unit that can portray this accuracy. The wider the available range, the finer the scale of attenuation coefficients that can be used. As mentioı ɜd above, modern scanners use a scale of -1000 to +3000, which corresponds to a 12-bit representation for each voxel.

Slice thickness affects the final image through the partial volume effect and statistical noise. The partial volume effect occurs when only part of structure falls within a slice, resulting in unequal irradiation from from angles. The partial volume effect produces streaks in the fnal image. Statistical noise affects the image as the slice is narrowed. Thin slices generally have better resolution than thick slices, so for this reason they are preferred. However, the narrower the slice, the lower the number of x-ray photons which are used to form the final image. As the number of photons decreases, noise in the system begins to overwhelm the image signal, so the resolution advantage gained by narrow slices is defeated.

Beam scatter affects the final image because the image reconstruction algorithms assume that the x-rays travel in a straight line from source to detector. Scattered rays alter the attenuation integrals obtained on each line, thereby reducing resolution and image quality. For this reason, proper collimation at both the X-ray source and detector is crucial.

The quality of the image is also affected by beam hardening. Beam hardening occurs as the result of using x-rays which encompass a range of energies instead of being monochromatic. As an x-ray beam transits a body, the lower energy x-rays are preferentially absorbed (attenuated) so that as the beam travels further, the average energy of the beam increases. This increase in average beam strength results in less absorption of the beam at a distance from the source than is absorbed by an equally dense structures close to the source. Beam hardening is inescapable at the present because x-rays sources which are capable of generating monochromatic x-rays do so at an energy which allows the beam to penetrate only a few centimeters into the patient rather than pass through to the detector. The perceived effect of beam hardening is that objects further from the x-ray source are less dense in the reconstructed image than they are in fact, resulting in streaks. Beam hardening effects are corrected for by employing iterative methods before performing final image reconstruction.

Image artifacts can be caused by the scanner or by the subject being scanned. The artifacts are classified as rings (caused by miscalibration, misalignment of detectors, or detector failure) and streaks (caused by patient motion, low sampling, or beam hardening). These problems can be prevented by a combination of CT operator actions and CT scanner design. In present generation CT scanners, their effects are negligible. Because these factors no longer present image reconstruction problems, the formation of high quality reconstructed images from a CT scanner is possible.

2.3 Ultrasound Tomography Scanners.

The introduction of X-ray Computed Tomography in the 1970's opened the door for the development of other high technology imaging techniques employing other forms of radiation. Computed Ultrasound Tomography, or simply Ultrasound, as described in [Bar81], [Fle84], [Gre82], [Hil86], [Kri88], [McD81], [Mey87], [Rob85b], [Tay85], [Wel81], and [Win84], was first demonstrated in 1973 by Greenleaf, and was one of the first of these new techniques to move into a clinical setting. Whereas CT scans form their images of the human body based on X-ray attenuation within a volume, Ultrasound uses the acoustical impedance properties of the tissues to form its images. Ultrasound systems operate by directing a short pulse of acoustical energy into a patient and then listening for the reflected or transmitted acoustical signal. The signals can be used to create a visual display which portrays the distribution of objects in the medium. The appeal of ultrasound lies in its low cost, portability, mm range resolution, and its use of non-ionizing radiation. The drawbacks associated with ultrasound come from its difficulty in imaging the heart and brain and the low contrast inherent in the image.

Ultrasound is defined to be any acoustical energy which contains frequencies higher than the audible hearing limit. In the diagnostic imaging context, ultrasound frequencies between 0.5 and 25 MHz are used, but the most commonly used frequencies are in the 1 - 15 MHz range. Diagnostic Ultrasound imaging seeks to determine the properties of portions of the human body by using the information obtained from observing the manner in which ultrasound waves are disturbed as they pass through the body. Figure 2.4 contains a simplified block diagram of a Transmission Diagnostic Ultrasound scanner depicting its major components. Reflection and Diffraction Diagnostic Ultrasound scanners differ in their placement of transducers and required transducer/body motion but employ the same basic components in a similar manner. The important difference between transmission systems and reflection/diffraction systems lies in the nature of the image. Transmission systems provide a reconstructed view of the imaged volume based upon the principles of tomographic reconstruction presented in the discussion on X-ray CT, whereas the other two systems do not.

The transmission ultrasound system operates as follows. Upon receipt of a signal from the data acquisition system, the transmitting transducer emits the ultrasound waves which pass through the patient's body and are received by the receiving transducer. When the pulse arrives at the receiving transducer a signal is sent to the data acquisition system where the time required for the pulse to traverse the body is computed. After each pulse, the transmitter/receiver pair moves along the line of translation to the next data acquisition point, where another elapsed time is obtained. The entire set of timings for one orientation of the line of translation provides the data for one projection. A set of projections is

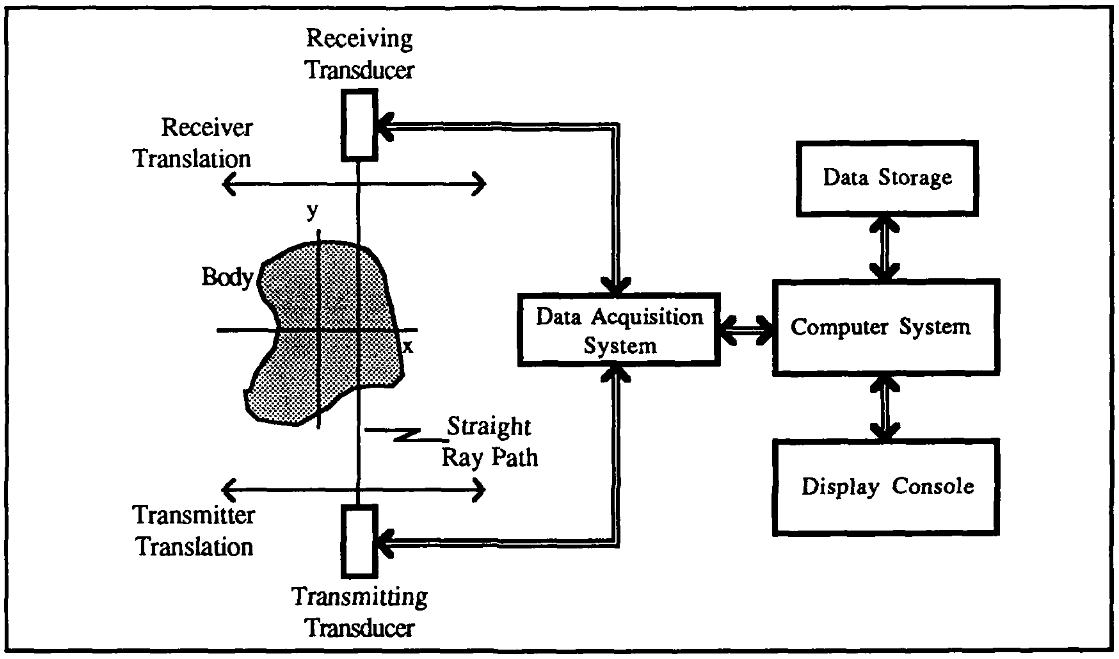


Figure 2.4: Transmission Diagnostic Ultrasound Scanner Diagram.

gathered by rotating the line of translation through small angular steps and repeating the transmitter/receiver translation process at each location. The complete set of projections is processed by the computer system to form the image of the interior of the body. The image can then be stored for later viewing or displayed at the display console.

Diffraction and reflection ultrasound do not employ tomographic reconstruction. Instead, the image is formed based upon the strength of the returned ultrasound signal and several display formats, A-mode[1], B-mode[2], M-mode[3], and C-mode[4], are available for visualizing the signal. For the purposes of three-dimensional display, the B-mode and C-modes are relevant. B-mode displays an image based on the amplitude of the returned echo at the receiving transducer. B-mode scanning can take place as either a real-time scan or a static scan, with the primary difference between the two techniques being the use of electrical drivers and circuitry to direct the ultrasound beam instead of manual positioning as in static scans. C-mode scanning provides a visual display of total attenuation of the ultrasound signal at each point in the image. The signal for the image is built up from the strength of the signal which has transmitted the body rather than from echoes.

The clinical Ultrasound picture is assembled from the strong echoes that occur at the organ interfaces and from the weak echoes that are produced by acoustical scattering within tissues. The respective uses for these echoes are to determine the gross anatomical layout of the imaged volume and to determine the textural characteristics of the tissues within the volume. The position of the source of the echo is determined based on the direction of the beam and the elapsed time from emission to detection. There are two broad categories of Ultrasound imaging techniques. The first category, represented by reflection tomography and transmission tomography, contains those reconstruction methods which employ the straight ray assumption using either reflected or transmitted ultrasound waves. The straight ray assumption proposes that ultrasound waves travel in a straight line from source to

---

[1]A-mode - amplitude mode, a direct display of echo amplitude versus distance into the tissue, a visual display of the reflectivity of the source and not its appearance.

[2]B-mode - brightness mode, a display of echo amplitude as brightness on a CRT, reconstructs a visual display of tissue appearance and not echo strength as in A-mode

[3]M-mode - time-motion mode, one intensity modulated A-line displayed as a function of time, a visual display of how the echo from one region changes over time. Employed in doppler ultrasound scans.

[4]C-mode - a map of total attenuation at each point that the ultrasound beam encountered. Does not employ echoes to form the signal.

detector without diffraction, refraction, or multiple reflections. The second category, known as diffraction tomography, takes diffraction of the ultrasound waves into account.

The first category of Ultrasound imaging techniques, transmission tomography, relies upon the straight ray assumption and involves taking measurements of transmitted acoustical energy. The straight ray assumption in Ultrasound simplifies the computational process by assuming that the sound waves travel in a straight line from source to echo point to detector. The typical transmission tomography configuration places the transmitting and receiving transducers on opposite sides of the portion of the body to be imaged. A data set for a single slice is obtained by taking acoustical signal measurements at small angular intervals as the transmitter and receiver are rotated around the object. The shape and characteristics of the object can be reconstructed from transmission delay measurements. Image reconstruction is performed using the elapsed time obtained for the different orientations of the transducers.

Reflection tomography obtains integrals of reflectivity from measurements of echoes scattered back towards the ultrasound source . This method has seen the widest clinical use. Like transmission tomography, reflection tomography relies upon the straight-ray assumption. Reflection tomography systems use a small acoustical energy emitter to transmit short duration pulses into a medium. The pulses are brief, one or two cycles, because the x-axis (axial) resolution depends on the brevity of the pulse. To limit the pulse, special crystals, such as lead zirconate titanate ceramic (PZT) are used and the back of the transducer crystal covered with a mechanical damping material. In order to achieve acceptable y-axis, lateral, resolution, divergence must be minimized. Divergence can be minimized by making the transmitter face large compared to the wavelength, but this limits the focal area of the transmitter. By changing transmitter faces, or using multiple crystals in the face and electronic focusing, the focus of the transmitter is changed and divergence is limited. The effects of divergence can not be completely eliminated because the beam divergence is also affected by the distance traveled.

The echoes returning to the emitter at any particular instant in time originate from acoustical scatterers located on the surface of the expanding spherical wavefront. The fraction of the incident beam energy reflected at the tissue interface depends on the the difference in the acoustical impedance of the two tissues at their boundary. The radius of each sphere is proportional to the elapsed time since the signal was emitted, and the resulting series of signals received from each ultrasound broadcast can be regarded as a projection. The image can then be constructed using convolution/backprojection techniques, described in [Bar81] and [Ros82], which take the one-dimensional output of the system and reconstruct an approximation of the two-dimensional space which was imaged.

The third computerized ultrasound technique, diffraction tomography, differs from the other techniques in that it attempts to compensate for the diffraction of the signal which occurs within the body. For the straight ray assumption employed in transmission and reflection tomography to be valid, the wavelength of the ultrasound signal must be smaller than any inhomogeneities present in the medium. However, because of signal attenuation considerations, wavelengths shorter than 0.3 mm cannot be used in the human body. Since structures smaller than 0.3mm exist within the body, the straight ray assumption may be violated. Significant ultrasonic signal diffraction effects have been demonstrated in the laboratory, hence the motivation for diffraction tomography.
Figure 2.5 contains a simplified representation of the diffraction ultrasound process. A single signal transmission using the geometry in Figure 2.5A is sufficient for estimating the spatial Fourier transform over all or part of the circle in Figure 2.5B. The data for the entire Fourier transform domain is obtained by taking multiple views of the object from different angles so that the circle in Figure 2.5B pivots around the origin of Fourier space. When all the views are completed, the Fourier domain data are interpolated onto a rectangular grid so that the fast Fourier transform can be used to reconstruct the image.

Ultrasound image quality is affected by several factors. First among these is the

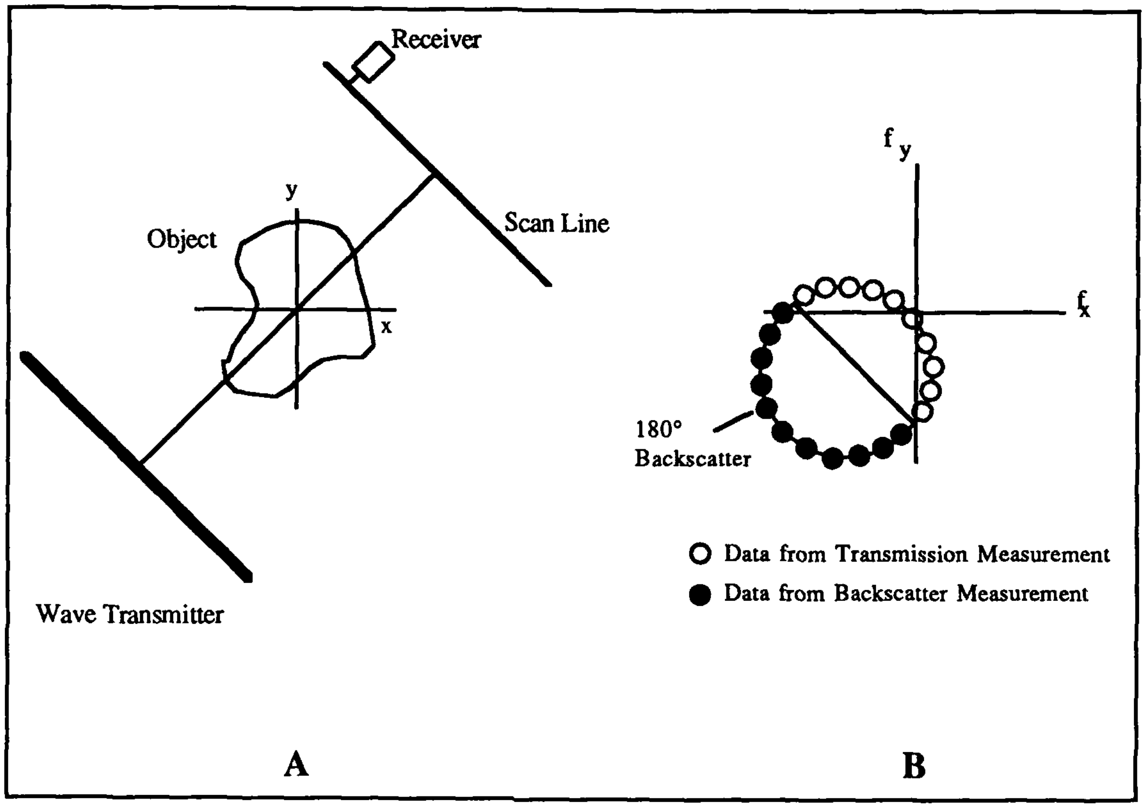

Receiver
Scan Line
Object
y
x
Wave Transmitter
A
f_y
f_x
180°
Backscatter
○ Data from Transmission Measurement
● Data from Backscatter Measurement
B

Figure 2.5: Diffraction Tomography Measurement and Reconstruction (based upon [Rob85b]).

fact that the size of the structures to be imaged is close to the minimum usable wavelength, so the ability to achieve high spatial resolution is severely limited. A further restriction on resolution occurs due to the increase in signal attenuation with frequency increase, forcing a compromise between tissue penetration and image resolution. The image quality effect of this compromise can be offset somewhat by varying the gain of the receiver based on the elapsed time between pulse transmission and reception; the greater the elapsed time, the higher the gain. However, this technique, called time-gain compensation, can not entirely offset the loss of signal which occurs at higher frequencies.

An additional factor which affects image quality are the underlying assumptions made concerning ultrasound wave propagation. Both categories of image reconstruction techniques rely upon simplifying assumptions concerning the behavior of the ultrasound waves as they propagate through tissue to make the image reconstruction problem tractable.

While the assumptions are good approximations of ultrasound wave behavior, these assumptions do not completely characterize the actual behavior of ultrasound waves in body tissues, leading to some inaccuracies in the reconstruction. Two assumptions which have a large impact on image quality are the assumption that ultrasound waves travel in either straight rays or along straight rays with direction changes caused only by diffraction and that the velocity of the wave is constant in all tissues. The straight-ray assumption is violated through beam-divergence, refraction, reverberation, and diffraction in the subject. Other than for diffraction, these effects can not be corrected for and must be recognized by the operator. The assumption of constant velocity for the ultrasound wavefront in all tissues results in misplacement of tissues since the x- and y- coordinates for an echo are computed based on the time-of-flight for the pulse. While the constant velocity assumption degrades the quality of the image, it, paradoxically, provides significant clinical information because tissue displacement can be used to characterize the type of tissue lying between the misplaced tissue and transmitter. The problems involved in measuring backscatter, determining the velocity of the ultrasound wave, and determining wave attenuation are active research areas.

Image quality is further degraded due to the fact that the propagation of ultrasound waves in tissue is a 3D process, but ultrasound machines measure and reconstruct data in 2D to simplify the image reconstruction process. Performing image reconstruction in 2D degrades the image even more because 2D reconstruction is sensitive to noise. As stated in [Rob85b], exploitation of the potential for ultrasound requires more accurate models of the wave propagation process and more sophisticated, possibly 3D, reconstruction methods. These factors affecting image resolution increase the resolution of a modern ultrasound scanner from the theoretical limit of 0.3mm at 25 MHz to about 1mm at that frequency.

Because the ultrasound image data is presented slice-by-slice, in a manner similar to that found in CT, each slice of an ultrasound image is represented using a 2D array. Each array element holds the gray-scale value for the corresponding position in the reconstructed image. The entire 3D volume imaged by the ultrasound scanner can be represented by a 3D array formed from the 2D arrays for each slice.

2.4 Magnetic Resonance Imaging.

Magnetic Resonance Imaging (MRI), also known as Nuclear Magnetic Resonance (NMR), is the newest of the 3D medical imaging modalities. The NMR phenomenon is based on physical principles which are well known. MRI uses proton density and the radiofrequency response of spinning atomic nuclei to magnetic fields and radiowaves to form an image. MRI systems operate by first imposing a strong magnetic field upon the body to be imaged and then exposing the body to radiowaves. In response, the spinning nuclei emit radiofrequency (RF) energy which can be detected by a receiving antenna. The picture formed in an MRI study portrays the aggregate RF response received from each voxel in the body that was imaged. The discussion is presented in terms of the nucleus of the hydrogen atom because it is the most abundant element in the body and yields the strongest MRI signal because its gyromagnetic ratio, γ, is the largest of any element and because it is the most abundant. The following material is based on [Axe83], [Bar88], [Bus88], [Byd88], [Hil85], [Hin83], [Kou84], [Mey87], [New83], [Old88], [Pyk82], [Rob85a], [Run84], [Sch87], [Sta88], [Wel81], and [Won87]. These books and articles provide a deeper understanding of the mathematical and physical principles underlying MRI than is provided here.

A MRI CT system has ten major components: the main magnet, the gradient magnet, the RF coil or probe, the RF waveform synthesizer, the gradient magnet waveform synthesizer, the data acquisition system, the microcomputer, the main computer, a data storage system, and a display console. Figure 2.6 presents a diagram showing the interrelationships of the major components of a typical MRI scanner. An MRI scan image is formed as follows. The patient lies on a positioning table completely within the bore of the main and gradient magnets and inside the RF coil. The main magnet provides the strong static magnetic field, H_0, necessary for MRI scans. Currently, over 95% of the main magnets in use are superconducting magnets. This preference is due to the strong field strength available, up to 2 Tesla [1], as compared to permanent and conventional magnets which are limited to about 0.3 Tesla. The strong magnetic field improves the signal-to-noise ratio of the device. The gradient magnet generates the magnetic field gradients along the x-, y-, and z-axis used for position determination. The RF coil transmits RF signals to and receives RF signals from the atomic nuclei in the patient. The main computer generates the RF and gradient waveforms and reconstructs images after data acquisition is complete. The microcomputer takes the RF and gradient waveform data

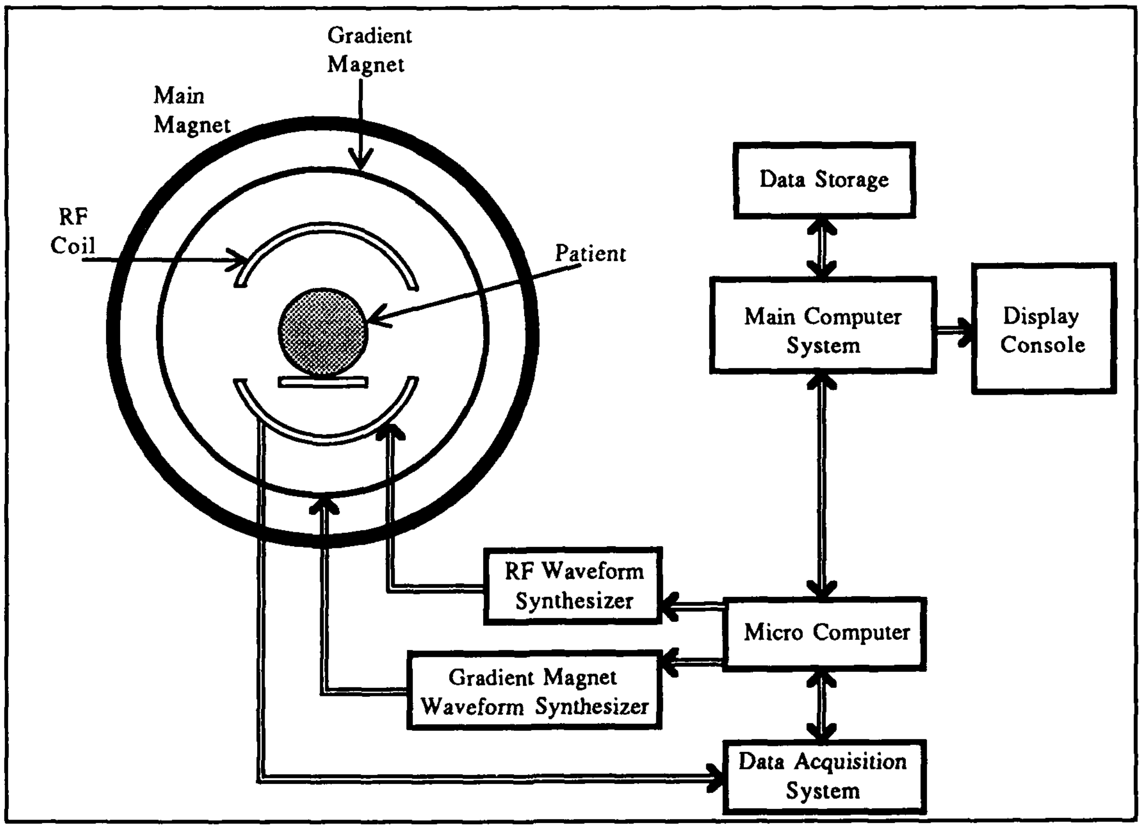


Figure 2.6: Magnetic Resonance Imaging Scanner Diagram.

[1]One Tesla = 10^4 Gauss, the strength of the Earth's magnetic field is 2×10^{-5} Tesla.

from the main computer and processes the waveform data for use by the waveform synthesizers. The gradient magnet waveform synthesizer takes the digital output from the microcomputer, converts it to an analog signal, amplifies the signal, and applies it to the x-, y-, and z-axis gradient magnets. The RF waveform synthesizer takes the digital output from the microcomputer, converts it to an analog signal, amplifies the signal, modulates it with the RF reference signal, and applies it to the RF coil. The response of the nuclei is detected by the RF coil/probe and transmitted to the data acquisition system, where the analog signal is converted to a digital signal, and then transferred to the main computer via the microcomputer. The main computer forms the image and can display it immediately or store it for future use. The heart of the process is the detection of the interaction of the magnetic fields and RF signal on the nuclei within the patient. This interaction is examined next.

Atomic nuclei which have an odd number of protons or neutrons possess a characteristic known as spin. The nucleus, being positively charged, generates a small magnetic field when it spins, and precesses about the axis of an externally applied magnetic field at a frequency which depends on the strength of the external magnetic field. In any given body, there is a large number of nuclei. If the body is placed in a strong magnetic field and then exposed to a short burst of RF energy at the frequency of the precessing nuclei, the nuclei precess in phase and emit a detectable, coherent RF signal at the precession frequency, called the Lamour frequency, w_0. The emitted RF signal frequency is directly related to the precession frequency, which, in turn, is directly related to the magnetic field strength. Given these relationships, the location of objects within the body can be determined by exposing the material to a strong, uniform magnetic field and then superimposing a weak magnetic field with a linear gradient to its field strength. Nuclei in different portions of the weak field precess at different frequencies, and so respond to different RF signals. Since the magnitude of the weak magnetic field gradient is known, the location of the nuclei which respond to a given frequency is also known.

In a MRI study, the RF transmitter is turned on to supply a quick burst of RF energy at a given frequency, and then turned off so that the resonate response of the nuclei at a specific location on the gradient can be detected. The nuclei at the desired position on the gradient precess at the same frequency as the RF pulse and resonate coherently at this frequency for a time. The RF signal emitted by the nuclei gradually fades as coherence is lost. The strength of the response and the time until the signal fades are governed primarily by the spin density, by two physical properties, the T_1 and T_2 relaxation times, and by T_2*, a property of the MRI equipment[1].

Spin density (SD) is a measure of the amount of free, or mobile, hydrogen that is available in the body for generation of the MRI signal. Hydrogen which is tightly bound within molecules, such as that found in bone, does not contribute to the MRI signal. Rather, the signal arises from the hydrogen which is mobile, ie., more loosely bound, which is the case for hydrogen found in fat and water. The spin density is not manipulated during MRI imaging, it sets the upper potential for signal strength. The RF pulse sequences used in MRI attempt to tap this potential by aligning the nuclear magnetic moments of the individual protons so that their vector sum results in a net magnetization vector of sufficient strength to be detected. The timing of the sequences and the strength of the signal are controlled by T_1, T_2, and T_2*

T_1 and T_2 are referred to as relaxation times because they describe the rate at which a substance loses the magnetic tension which makes their RF response detectable. T_1, also called the longitudinal or spin-lattice relaxation time, describes the amount of time required for the energy injected into the nuclei during the RF pulse to fade away, or, equivalently, T1 is the time required for the protons to return to thermal equilibrium with their

[1]Other components of the image such as chemical shift, RF pulse susceptibility, RF pulse absorption, effects arising from the flow of fluids within the body, and perfusion have been investigated. These factors weigh more heavily in MR spectroscopy than in imaging, so are not addressed here.

surroundings by re-aligning with the strong external magnetic field. In clinical use, T_1 describes the amount of time which must pass before the next RF pulse can be applied and a measurable signal obtained. The T_1 time depends on the molecular environment of the tissue, and can be used to differentiate between various tissues. The phenomenon can be explained as follows. Magnetization occurs because nuclei tend to align themselves along the magnetic lines of force when placed in a magnetic field. The protons point either north or south, with a slight majority of the protons pointing north, since that direction has a lower energy level. This slight net northward orientation results in the magnetization of the tissue. When protons are exposed to an external magnetic field, magnetization occurs, rapidly at first but then gradually slows down as the maximum magnetization for the material is approached. The increase in magnetization of a material with time is represented by an exponential time constant called T_1. Since being magnetized is the equilibrium state in MRI, T_1 not only describes how long it takes to return to equilibrium following RF excitation but also the amount of time that must elapse before the first RF signal can be applied. Since different materials, and different locations within the magnetic gradient, have different T_1s, the RF pattern can be varied to enhance image contrast. Whereas T_1 describes the amount of time which must elapse before the next RF signal can be applied, T_2, also called the spin-lattice or transverse relaxation time, describes the amount of time that the signal from the last RF pulse remains detectable. Detectability is related to the number of protons which are precessing in phase within the slice, so an equivalent characterization of T_2 is that it is the time required to de-phase the precessing the magnetic moments of the protons to the point where there is no detectable signal.

Table 2.1 presents average T_1 and T_2 values for normal organs within the human body. T_1 and T_2 relaxation times are characteristic properties of all substances and, as can be seen in Table 1, can be used to differentiate them in the manner similar to the way in which X-ray attenuation coefficients differentiate materials in a CT scan or acoustical properties differentiate materials in an Ultrasound study. The ability to differentiate materials is further exemplified in Table 2, from [Mor86]. Table 2.2 presents average T_1 relaxation times for organs in the body, note that even normal and tumorous tissue of the same organ have different T_1 times, which allows use of T_1 times to differentiate between normal and diseased organs in the body. T_2* is not a relaxation time, but is instead determined by the equipment used to perform MRI imaging. T_2* is defined to be the time it takes a real MRI signal to disappear. This time is related to the quality of the magnets used to impose the magnetic field, the larger or more numerous the inhomogeneities in

TABLE 2.1

T_1 and T_2 Relaxation Times for Body Tissues

TISSUE	T_1 (msec)	T_2 (msec)
Fat	170	80
Liver	250	50
Brain - white matter	350	80
Brain - gray matter	500	100
Spleen	450	75
Kidney - cortex	340	70
Muscle*	400	50
Blood	720	175
Cerebrospinal Fluid	1500	260
Pancreas*	290	60
Water	2500	2500

the field generated by the magnet, the shorter the T_2*. The significance of these times is that they control the time interval between successive RF pulses, the image quality, the contrast between tissues, and the amount of time that the RF signal is present after the pulse is applied.

* denotes data from [Mor86] at 8.5 MHz, otherwise data is from [Els86] at .15 Tesla.

Evidence indicates that variations between healthy and diseased tissues are more closely related to changes in relaxation times than to changes in protcn densit,. RF pulse timings can be manipulated so that the relative contributions of T_1, T_2, and proton density to the final image vary, thereby enhancing selected details in the body and providing a more clinically useful picture. Three important RF pulse sequences are saturation recovery, inversion recovery, and spin-echo. Each method requires different timings of the RF pulses, and the image quality is affected by different factors for each sequence. A comprehensive examination of these techniques is beyond the scope of this paper, further information can be found in [New83].

TABLE 2.2

T_1 Relaxation Times for Healthy and Diseased Organs

ORGAN	T_1 (msec)
Normal Liver	140 - 170
Liver - Cirrhosis	180 - 300
Liver - Cancer	300 - 450
Brain - normal white matter*	290
Brain - normal gray matter*	525
Brain - Benign Tumor*	520 - 600
Brain - Cancer*	750 - 1520
Brain - Metastatic Cancer*	510 - 1370
Kidney - normal cortex	300 - 340
Kidney - Cancer	400 - 450
Normal Prostate	250 - 325
Prostate - Cancer	350 - 450
Normal Pancreas	180 - 200
Pancreas - Cancer	275 - 400

* denotes data at 6.5 MHz, otherwise data is at 1.7 MHz.

The different methods that have been developed for forming an MRI image are point, line, planar and whole volume. The point and line techniques are not used clinically due to the length of time required to obtain an image comparable in quality to planar or whole volume techniques, therefore the discussion is restricted to planar and whole volume imaging. Before turning to this discussion, the use of phase encoding and frequency encoding for retrieving spatial information in the returned RF signal must be explained. Phase encoding is used to separate the voxels in one direction, say along the y-axis, and frequency encoding is used to separate the voxels in the other direction, say along the x-axis. Phase encoding is created by imposing a gradient in the magnetic field over a given volume, for this example, the y-direction. Initially, the protons in the slice all have the same amount of magnetization and are in phase because of the magnetization imposed by the main magnet. When the gradient magnet is turned on, the strength of the field increases in the y-direction, which forces the magnetization vectors of the protons out of phase. The amount of phase shift a given proton experiences depends on its position in the vertical direction. When the gradient magnet is turned off, the protons are again within a homogeneous magnetic field. The protons all precess at the same frequency but the phase shift induced by the gradient magnet remains. The protons still emit RF signals of the same frequency, but with different phases depending on their location along the y-axis. Figure 2.7 depicts phase encoding over a 4 x 4 voxel volume, where the angle between the two arrows represents the phase angle. Frequency encoding separates the protons in the other, x-axis, direction. Frequency encoding is accomplished by turning the gradient magnet on when the RF signal is produced. If the gradient is applied along the x-axis, within this slice the protons can be differentiated by their x-position because the protons at a given x position respond to different RF signals. The result of these two applications of the gradient magnet is that the x- and the y-coordinates of each proton can be determined.

The planar technique produces images based on signals arising simultaneously from all points in a plane. The technique uses a single phase-encoding setting for the transverse slice and multiple frequency-encodings to select each row of the plane. An image of a given transverse slice is obtained by using a slice-selection pulse to excite all the nuclei within that plane of the body. Two techniques commonly used to select the plane are selective-irradiation and oscillating-gradients, both techniques use a magnetic field gradient in the z-direction to select the slice to be imaged. The selective-irradiation technique uses a magnetic field gradient which does not vary with time, and excites only those the nuclei which are resonating within the narrow range of frequencies which define the plane to be imaged. The oscillating-field approach uses a magnetic gradient which periodically reverses during the plane selection, and so only those nuclei which lie within the plane where the field does not reverse have a sufficiently strong signal to form an image. To accumulate the data for the slice, the phase-encode, frequency-encode, slice-select sequence is repeated at least once for each of the rows in the phase-encoded direction. Each excitation defines a new line onto which the plane being imaged projects. The process is analogous to rotating the gantry around the patient when performing a CT scan. The end result is that projections of the plane are collected at various angles around the patient. When sufficient projections are accumulated, the computer can reconstruct the image of the plane. The image reconstruction is performed by conducting a Fourier transform on each of the data lines, and then backprojecting the result to form the image. A complete planar MRI scan, or multi-slice image, produces a set of contiguous slices of the patient's body and can be used for 3D imaging.

The whole volume technique is an extension of the planar technique. In this technique, the slice-selection pulse is replaced by a RF pulse which selects the entire volume of interest. The imaged volume is divided into slices during the reconstruction phase using the RF signals which were encoded with spatial information in the slice-selection direction. As in planar imaging, phase-encoding is used to provide spatial information, except in this case, phase-encoding is used for both the slicing direction and along one other axis. Frequency encoding is not used since the entire volume of tissue

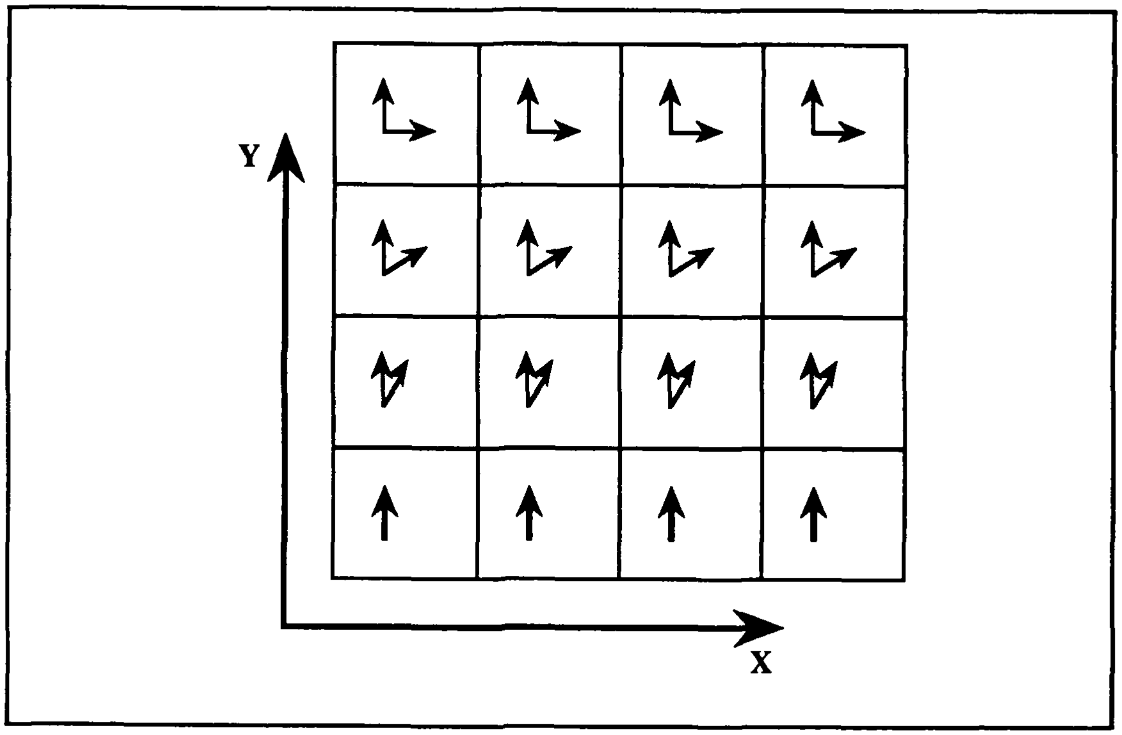


Figure 2.7: Matrix of Phase-encoded Protons (from [Sta88]).

must respond. Volume imaging creates thinner contiguous slices than planar imaging but takes significantly more time.

The quality of the images formed in MRI is primarily affected by the homogeneity of the main magnetic field and the gradient fields. Homogeneity of the main magnetic field is achieved through stringent fabrication techniques and the use of shim coils for the local adjustment of the magnetic field along the length of the bore of the main magnet. The more homogeneous the fields, the stronger the signal which is produced, resulting in a better signal-to-noise ratio. Quality is also influenced by the RF coil design as it controls the sensitivity of the scanner. The capability for positioning of the RF probe close to the patient and uniform broadcast intensity are important design parameters which impact image quality. Because of the desirability of uniformity and close proximity, special purpose RF probes have been designed for skull and body part imaging and are used in addition to the main RF coil in the MRI system. In the case of body part imaging, the pliable RF probe design allows direct contact with the portion of the body to be imaged.

The recent development of paramagnetic contrast media offers the potential for improving contrast in T_1 and T_2 weighted images with a concomitant improvement in diagnostic capability for the technology.

Because of the interrelated effects of the T_1, T_2 and mobile proton density, there is no simple physical interpretation of the MRI image. Proper interpretation of the image requires knowledge of the magnetic field intensity, pulse timings, image enhancement techniques used[1], and the portion of the gray-scale being viewed. A discussion of image interpretation is beyond the scope of this paper. Since planar MRI image data is presented slice-by-slice, in a manner similar to CT, each slice can be represented using a 2D array. Each array element holds the gray-scale value for the corresponding position in the reconstructed image. Current MRI technology supports the construction of 256 x 256 images with a resolution of approximately 1mm on a side. The entire 3D volume can be represented by a 3D array formed using the 2D arrays for each slice, but interpolation is not required since the slices cover the entire volume of the patient that was scanned. In volume MRI imaging, the volume is represented using a 3D array with the values in each array position corresponding to the signal received from the corresponding position within the volume of the patient. Resolution is lower and voxel dimension is greater than for the planar technique, the chief asset of this technique is its relatively high speed when compared to planar imaging.

[1] Image enhancement techniques include, but are not limited to, the use of paramagnetic contrast agents, different image reconstruction algorithms, and eigenimage filtering.

2.5 Single Photon Emission Computed Tomography.

Building upon technology developed in the late 1940's and early 1950's, Single Photon Emission Computed Tomography (SPECT) was pioneered by Kuhl and Edwards in their 1963 paper [Kuh63] describing emission tomographic radionuclide brain-imaging. The development of the Technitium-99m generator and the Anger gamma camera (1967) along with ready availability of computing power adequate to the image formation task allowed this technology to take-off in the early 1970's. Currently, SPECT is an established technology with widespread clinical use. Its attractiveness comes from its use of biologically active radioactive tracers which allow study of body organs by physiologic activity such as blood flow or oxygen transport volume. This biochemical activity imaging capability complements the anatomical imaging capability of MRI, CT and Ultrasound. The material for this section is drawn from [Axe87], [Bai87], [Bud80], [Cha87], [Cro86], [Del84], [Ell82], [Eng86], [Ess84], [Gus85], [Hal88], [Hob88], [Jas80], [Jas88], [Lit83], [Mue88], [Mur80], [Sor87], [Tod83], and [Wel81].

The primary goal of SPECT imaging is to provide an accurate series of 2D images which portray the 3D distribution of the radioisotope within the organ(s) of interest as a function of time. The images provide the means for quantification of source density per unit volume and for visualization of source distribution. These capabilities permit SPECT to track movement of fluids in the body, metabolism, and drug concentrations. The greatest advantage of SPECT over planar radioisotope imaging methods lies in its ability to remove the superposition of organs. The radioisotopes used in the procedure are relatively long-lived γ emitters, those in common clinical use are Technitium-99m, Iodine-123, Xenon-133, Thallium-210, and Gallium-67. These radioisotopes are tagged to biologically active compounds and then injected into the patient. The biologically active compounds are selected for their affinity for the specific organ to be imaged. A disadvantage of SPECT imaging, which is overcome by Positron Emission Tomography, is that the use of radionuclides alters the biochemical activity of the tagged compound. Therefore, the uptake rate and organ specificity of the tagged compound is not identical to that of the untagged compound. In any event, the selectivity of the tagged biochemical for the target organ leads to an accumulation of radioactive material in the target organ. Eventually, a sufficient dose is gathered for imaging to be performed. The result of the imaging procedure is a picture which graphically portrays the concentration of the radiopharmaceutical within the organ.

The SPECT imaging system is completely dependent for its functioning upon the decay of radioactive nuclides to stable states. Because of its importance to the functioning of the modality, a brief overview is presented, further information may be found in [Eva85], [Kap63], and [Kno79]. Radioactive decay can occur as either electromagnetic radiation or charged particle emission. There are three processes through which a radionuclide attains stability, these are alpha (α) particle decay, beta (β) particle decay, and *gamma (γ) decay. In alpha decay, a radionuclide emits a heavy charged particle, the* nucleus of a helium atom, at a discrete, fixed energy level along with photon emissions. Alpha-emitting nuclides are not used in SPECT imaging, or any other diagnostic procedure, because of the health hazard they pose. Beta decay is important to positron emission tomography imaging and is discussed there. Gamma decay is the backbone of SPECT imaging. Gamma decay can take one of two forms, emission of a high-energy photon or through electronic conversion, but only the photon emission process is relevant to SPECT imaging. In high-energy photon emission, the excess energy in the nucleus is released as a γ ray. A γ ray and an x-ray of the same energy interact with matter in the same manner, the only difference between the two being that an x-ray is emitted by transitions of electrons through energy states and γ rays are emitted by the nucleus as it changes energy states.

The rate of radioactive decay is expressed in terms of disintegrations per second, the unit of radioactivity is the curie (Ci), which is defined as the radioactivity of a sample decaying at the rate of 3.7×10^{10} disintegrations/second. The curie is being replaced by the becquerel (Bq) as the international standard unit of measure, and is defined as the radioactivity of a sample decaying at the rate of 1 disintegration per second. The radioactivity of a sample depends on the amount of radioactive material present and the decay constant, λ, defined as the probability of decay per unit time for a single atom, for the material. The radioactivity per unit mass of a radionuclide is known as the specific activity of the sample and is expressed as milli-curies/microgram. The knowledge of specific activity is important because it relates directly to image quality.

Radioactive decay proceeds at an exponential rate, and the radioactivity of the sample follows the same course. The radioactivity of a sample at time t is given by: $R_t = R_0 * e^{-(\lambda t)}$, where R_0 is the initial radioactivity of the sample. The decay constant is determined experimentally by finding the half-life of the substance, $T_{1/2}$, and then calculating λ using $\lambda = 0.693 * T_{1/2}$. For medical imaging purposes, the biological half-life, defined as the rate of elimination of a biochemical from the body, must also be known, and is exponential. Knowing the radioactive half-life and the biological half-life allows the computation of the effective half-life, which is the sum of the radioactive and biological half-lifes. The effective half-life of the compound is important because it determines the dose which must be administered to achieve a given image resolution, or conversely, it determines the resolution achievable from a given dose. The basis for this relationship lies in the statistical nature of radioactive decay.

The probability that a given number of disintegrations, N, occur in a radioactive sample in time t is a Poisson distribution around the mean number of disintegrations, $\tilde{N}$, for the sample in time t. In practice, the Poisson distribution is approximated by the Gaussian distribution. The error in the radioactive measurement is expressed as percent standard deviation, %S.D., which is defined as %S.D. = $\frac{\sigma}{\sqrt{N}}$ where $\sigma = \tilde{N}^{.5}$. The precision of a given radioactive measurement can be increased by increasing the observed number of disintegrations, but to decrease the percent error by a factor of ten, the number of counts must be increased by a factor of 100. The precision of the image can be improved by either substantially increasing the dose to the patient or increasing the duration of the scan. However, since the effective and radioactive half-lifes are exponentially decreasing with time, the dose can not be safely increased to the point that it would offset the loss in resolution or precision. Realistically, the patient can not be expected to lie immobile for long periods of time. The best solution to this problem lies in increasing camera sensitivity, but this reduces image resolution. Due to dose limitations, patient movement, and the need for spatial resolution the %S.D. is in the 10 - 20% range in modern systems. This range for the %S.D. decreases the confidence in the observed result, as SPECT seeks to portray biochemical function by radioactive activity, and there is significant uncertainty in the results gathered by the modality.

The relationship between radioactive activity and biochemical function is based on the concentration of radioactive material evidenced within each voxel. The activity can be computed using image reconstruction techniques based on the number of radioactive counts gathered at each location as the gamma camera rotates around the patient. Once the reconstruction is performed, a 3D map of radioactive activity per voxel is at hand. Since γ is known, the number of radioactive atoms and their mass can be calculated for each voxel. At this point the %S.D. comes into play, as it decreases the accuracy of the activity distribution map, and hence the accuracy of the representation of the amount of radioactivity in each voxel.

For diagnostic purposes, the effect of the uncertainty is small if the concentration of the radionuclide varies smoothly within the sample volume. However, in the case of "hot" and "cold" radioactive lesions the effect is noticeable, and this problem has received the bulk of research attention. The difficulty in imaging "hot" and "cold" lesions comes from the inverse-square relationship between image resolution and statistical precision. Increasing the resolution of the image requires an increase in the number of voxels and in the size of the data collection matrix. However, for a constant number of counts, an increase in the size of the data collection matrix results in fewer counts per volume, thereby decreasing the statistical precision of the counts recorded for each voxel. Increasing the statistical precision by increasing the voxel size results in lower resolution. The straightforward solution to this problem, increasing the number of counts, can not be used for the reasons described above. Therefore, systems aim at achieving a balance between these two competing requirements. The common solution is to use a 64 x 64 data collection matrix and accept the 10 - 20 %S.D. Experimental systems have been described which use a 128 X 128 data collection matrix with no loss in statistical precision.

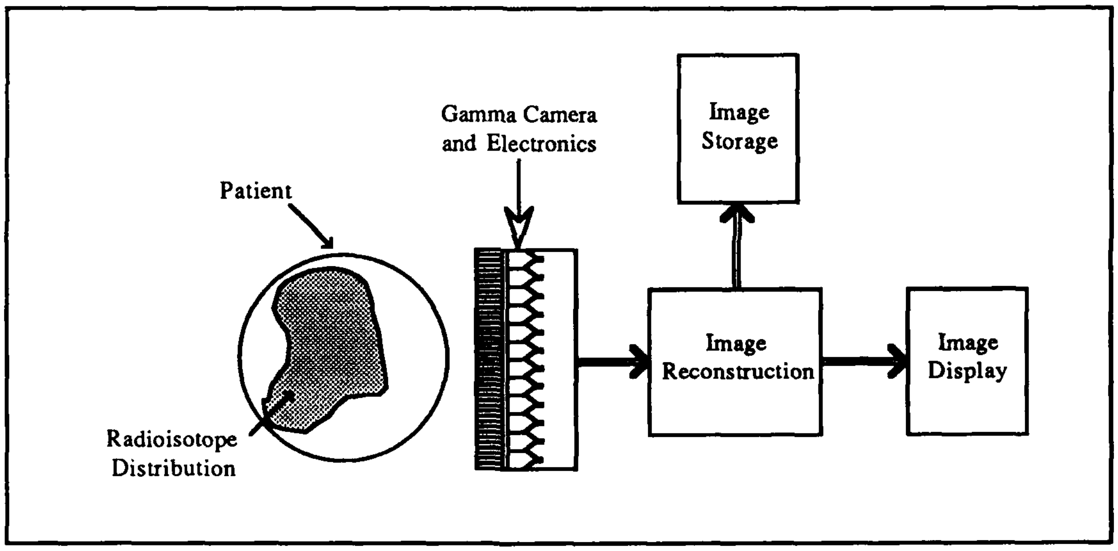


Figure 2.8: Single Photon Emission Computed Tomography System Diagram.

The important hardware design goals for any SPECT system are provision of high sensitivity with high resolution. The approaches taken to achieving these conflicting design goals have focused around varying collimator designs and multiple cameras. The collimator approaches have been those which use multiple pinholes, time varying apertures, and Fresnel coding. The multiple camera concept has been implemented by using camera layouts, which place a series of cameras around the patient which scan the patient using only lateral motion, and the rotating Anger camera approach. Due to its widespread use, its relatively good image quality, and the similarity of the basics of this system to the other methods, the rotating camera approach is described.

A block diagram of a typical SPECT system with rotating Anger gamma camera is presented in Figure 2.8, and a diagram of an Anger camera is in Figure 2.9. The major components of a SPECT system are the analog signal electronics for pulse amplification, the digital electronics for γ ray coordinate computation, the image reconstruction system, the Anger gamma ray camera, the collimator, and the radioactive nuclide within the patent. A SPECT image is formed by the detection of emitted γ-ray photons and the reconstruction of the image based on the coordinates computed for the photon emitter.

The components of the system work together as follows. The radioactive nuclide, which is concentrated in a body, undergoes a radioactive decay which results in the emission of γ photons. The photons pass through the body, and some of them strike the front of the gamma camera, which is the only unshielded portion of the camera. A gamma camera, pictured in Figure 2.9, uses a single large scintillation crystal viewed by photomultiplier tubes and using analog circuitry to determine the location of the gamma ray with the crystal. The camera is designed to detect photons in the 60 - 300 keV range, with the exact range determined by the collimator. Typically, an installation has two collimators; one designed to pass γs in the 80 - 200 keV low-energy range and a medium-energy collimator designed for up to 380 keV. The collimator is a key element of the system, since the arrangement of the holes determines the projection view of the radionuclide distribution and, because it determines the visible area of the detector, the sensitivity of the system. The holes in the collimator are typically straight, evenly spaced, and parallel, although other designs have been proposed and found clinical acceptance. The photons which arrive at the collimator at an angle such that they can pass through the holes in the collimator and reach the sodium iodide scintillation crystal. The γ photons interact with the iodide ions in the crystal through photoelectric or Compton scattering, forcing the iodide ions to lose electrons. The liberated electrons interact with the crystal lattice to produce light. The

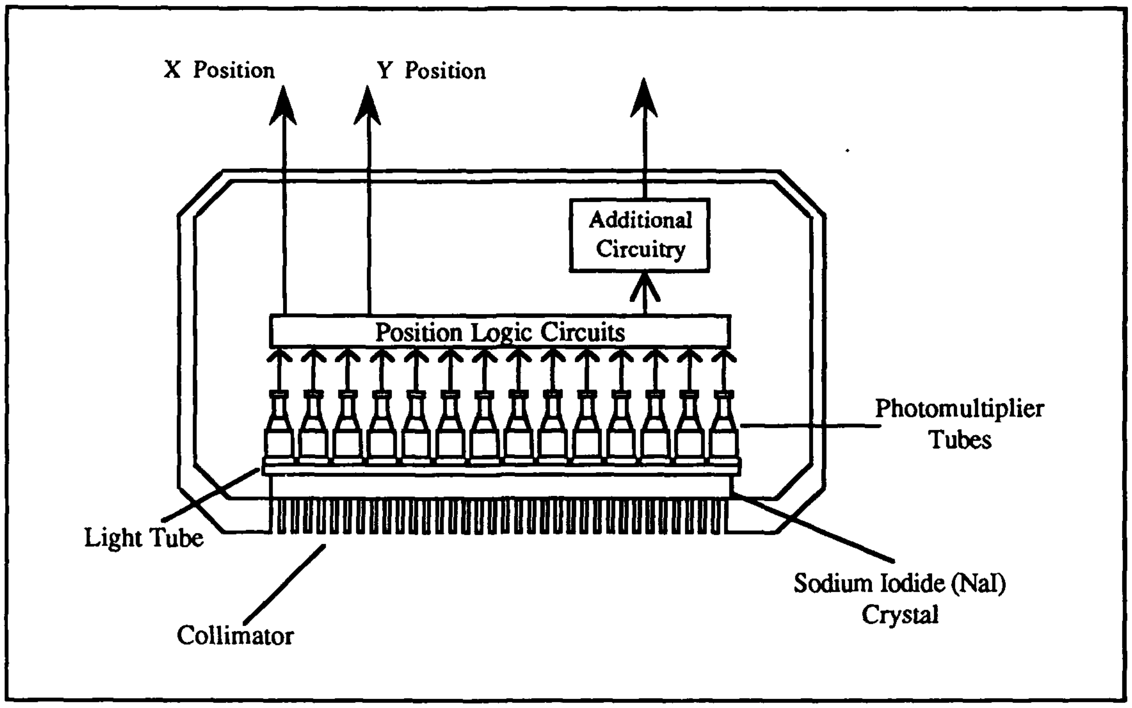


Figure 2.9: Anger Gamma Camera Diagram.

amount of light produced is directly proportional to the amount of energy in the γ photon. The position of the origin of the light in the crystal corresponds to the site within the crystal where the gamma ray interacted with the iodine. The amount of light emitted by the interaction of the γ photon with the crystal is very small, so this signal is not used directly to determine the location of the interaction. rather, the light is converted to an electrical signal and amplified by photomultiplier (PM) tubes placed directly behind the crystal. The PM tubes are closely packed and attached to the crystal. The PM tube absorbs the light from the crystal on its face and generates a low voltage. The dynodes in the subsequent amplification stages increase the signal by a factor of 10^6. This weak electrical charge emerges from the gamma camera and is processed in the analog-to-digital converter where it the electrical signal is amplified and digitized for processing by the image reconstruction system. Each photomultiplier tube receives a fraction of the light from the γ ray/crystal interaction proportional to its distance from the light source. The relative strengths of the electrical signal from each PM are used to determine the x and y coordinates of the source of the γ ray. A typical Anger camera is capable of between 20,000 and 100,000 interactions per second. This count rate limitation puts an upper limit on the sensitivity of the system, and hence on the accuracy of the reconstruction. To acquire a 3D image, the gamma camera is rotated around the patient at regular time intervals until a series of 64 x 64 matrices have been compiled for 60 to 100 angular positions, for a total of 300,000 - 400,000 disintegration counts. Once the examination is completed, the 3D volume is reconstructed.

The image quality issue must be approached from two points of view, the statistical significance of the results obtained and the final resolution of the image. As the statistical aspects of the image were reviewed previously, image resolution is addressed here. Final image resolution of 2-3mm for brain tissue imaging and 4-5mm for body imaging have been reported in the literature, but a clinical imaging systems' capability is somewhat poorer, being in the 9mm range for body imaging. Image resolution has several components, the chief ones being the degree to which scattered γ rays are rejected, the resolution of the collimator, and the intrinsic resolution of the scintillation camera (NaI crystal). Scatter rejection attempts to detect the γ rays which where scattered within the body by the amount of energy in the detected γ rays. The degree to which scattered γ rays are rejected is determined by the energy window selected for the system, in general, the energy of a γ ray decreases due to scattering. Scattered ray rejection is accomplished by setting a energy window within the system low enough to discriminate between unscattered and scattered rays. However, the attempt to discriminate based on energy level results in a tension between resolution and statistical significance because a wide window improves the statistics for the image. This improvement results from the detection of more of the γ rays which were emitted at the characteristic discrete energies of the radionuclide. Unfortunately, the band of the characteristic emission energies overlaps the energies found in the scattered γ rays, so an improvement in image statistics comes at the expense of resolution, and vice-versa. The final window setting results from a compromise between providing acceptable resolution and acquiring a statistically significant number of counts per pixel. Since scatter rejection by single window energy discrimination is incomplete for multiple energy γ ray emitting isotopes, techniques employing multiple windows, offpeak window selection, and energy weighted acquisition have been investigated. The multiple windows and offpeak window selection techniques employ some form of energy discrimination to accomplish scatter rejection. Energy weighted acquisition accepts all detected photons into the system, so no explicit rejection is performed. Instead, probabilistic weightings are assigned to the different energy values which may be detected and each signal is weighted by the probabilistic chance that the detected γ was not scattered.

The resolution of the scintillation camera is degraded with increased thickness of the NaI crystal, and improved with an increase in γ ray energy. The resolution of the collimator is affected by the collimator length, the diameter of the holes in the collimator, and the distance from the collimator face to the radiation source. Unfortunately, the resolution of the system is closely tied to the sensitivity of the system. Sensitivity is defined as the ratio of counts recorded by the crystal to the number of photons which reach the crystal. As in the case of statistical precision, resolution and sensitivity have an inverse-square relationship. So, for example, while an increase in the thickness of the NaI crystal reduces resolution, it improves sensitivity and an increase in γ ray energy decreases sensitivity but improves resolution. Since sensitivity and statistical precision are also related, a doubling in the resolution by an increase in γ ray energy results in a 16-fold decrease in statistical precision.

Efforts aimed at improving resolution without adversely affecting precision or sensitivity have taken two approaches, allowing the collimator to move closer to the patient and increasing the number of detectors in the system. Close collimator approach can be achieved by automatic patient contour tracking, changing the geometry of the edges of the collimator, and by changing the collimator itself. As an example of this last technique, a collimator with its holes angled at 30 degrees to the face is effective for close-range imaging of the heart and cranium. Proposals for increasing the number of detectors in the system have assumed two forms, either adopting the CT scan slice-by-slice technique or using multiple rotating cameras. Systems with two or three rotating Anger cameras have proven effective as they improve image quality across the board, with their cost being the increase in computations required to combine information from separate sources to form the image. Non-rotating, slice-by-slice systems, such as the Cleon-710 and Cleon-711, require multiple slices to form the 3D image. These systems improve image quality by increasing the number of counts collected through the use of several collectors (10 - 12) while maintaining statistical precision and sensitivity.

2.6 Positron Emission Tomography.

The first modern Positron Emission Tomography (PET) machines were described in 1973 by Brownell and Burnham [Bro73] and by Robertson [Rob73]. The roots of this technology go further back, however. The use of positrons for coincidence detection imaging was suggested by Wren in 1951, and later by Brownell and Sweet in 1953. This initial effort was followed up nearly a decade later by Robertson *et al* in 1962. Their procedure accomplished transverse section imaging using emitted positrons by using arrays of sodium iodide detectors. These initial efforts were limited in their ability to form a clinically useful image by the large number of scattered gamma rays which were included in the image (thereby reducing the resolution), poor reconstruction algorithms, and a limited number of angular views. The development of the technology which made CT possible also served to launch PET imaging. By 1975, when Phelps and Ter-Pogossian described their machine, the basics of modern PET scanning: Fourier reconstruction, high sampling rates, and attenuation correction, were in p[illegible]. PET imaging bears some resemblance to SPECT imaging, but the differences between the two give PET better image quality, albeit at somewhat higher cost. Its advantages are that it uses true tracers and not tags on pharmaceuticals, has greater resolution and sensitivity than SPECT, and possesses better quantitation[1] of radionuclide concentration than SPECT. The disadvantages of PET are its cost, the need for a cyclotron on-site raises the initial start-up costs to approximately $4 million, and the fact that most applications are still in the research phase. Material for this section is drawn from [Cha87], [DelG87], [DeP77], [Goo80], [Hob88], [Jas88], [Lam85], [Mue86], [Mue88], [Mur80], [Nic86], [Paa85], [Phe85a], [Phe85b], [Red81], [Rol82], [Sch86], [Sor87], [Sou83], [Ter85], [Tod83], [Wel81], and [Yam85].

As with SPECT, PET provides an image which quantitatively portrays the 3D distribution of a radioisotope tracers within an organ of interest. The radioisotopes used in this procedure are the short-lived positron emitters ^{15}O, ^{11}C, ^{13}N, and ^{18}F. However, rather than attaching the radioisotope to a biologically active pharmaceutical, the radioisotope is incorporated into the chemical structure of the pharmaceutical, thereby labeling the material without altering its biological activity. PET has demonstrated its capability for performing blood flow measurement, computing cerebral blood volume, and determining the metabolic rate of different compounds, such as glucose. In addition to these capabilities for measuring fluid movement and metabolism, PET has the capability for measuring regional distribution of pharmaceuticals, *in vivo* brain tumor typing, and for detection of pharmaceutical binding sites within the body.

PET uses the release of β^+ (beta) particles from the decaying radionuclide in a process known as β^+ decay. A β^+ particle, also called a positron, is identical in all respects to an electron except that it has a electric charge of +1 instead of the -1 electrical charge of the electron, therefore the positron is the antiparticle of the electron. In β^+ decay, a proton in the nucleus is converted into a neutron and the excess energy is emitted from the nucleus as a positron and a neutrino. The neutrino passes through the body and scanner without any interaction, so it does not play any part in forming the image. The β^+ particle is the

[1]Quantitation is defined as the process of making measurements on organs and organ systems

key imaging element in β^+ decay. After its release from the nucleus, the β^+ particle is decelerated through the drag exerted upon it by the surrounding negatively charged electrons. When the β^+ particle has dissipated enough of its energy so that it is in equilibrium with its surroundings and essentially motionless, the attractive force between the positively charged β^+ particle and an electron causes the formation of an electron/positron pair, which then annihilates converting all of its mass into energy. This liberated energy is realized as a pair of 511 keV γ rays which depart the scene of the annihilation traveling in opposite, or nearly opposite, directions. Because it is the location of the β^+ decay, and not the annihilation, which is desired, the resolution of a PET system is limited by the range of the β^+ particle, which is approximately 0.15 cm in human tissue. The practical resolution of the system is higher because it is limited by the number of detectors and their size. The production of two γ rays is the main reason for the increased resolution of PET over SPECT because of the localization of the annihilation event, and thereby the location of the decayed nucleus, which is achieved through the use of coincidence timing.

The importance of gathering adequate statistics for image formation is as important for PET imaging as it is for SPECT imaging. The quality of the statistics gathered for a PET scan is determined by its ability to detect and reject those γ rays which arrive at the detector that are not the product of the same annihilation event. The fact that only 20-30% of the γ rays reach the detector without being scattered highlights the importance of meeting this requirement in order to form diagnostically useful images. If a large percentage of scattered γ rays are accepted as coincident, both the image statistics and the resolution fall off. As there is no way to predict when a decay will occur and two γ rays released, a PET system relies on coincidence detection to determine if two gamma rays are the product of the same annihilation event. Coincidence is determined by setting a time window for the system, and any two γ rays arriving at opposing detectors within the time limit set by the window are deemed to be the product of the same event. The ability to reject γ rays which seemingly coincide is determined by the temporal resolution available in the system; the narrower the window, the better the system is able to reject false coincidences. Because some false coincidences are accepted by any PET system, lengthy (30 minutes or more) data acquisition times are used so that the number of true coincidences provides the desired image quality in terms of resolution and signal-to-noise ratio.

To date, there have been three classes of PET machines produced, the camera based machines, the Multi-wire Proportional Chamber machines and the multi-detector machines. The camera based machine closely resembles a SPECT system, with the only observable difference being that it has two opposing gamma cameras. The achievable image quality for the camera system is somewhat lower than for the multi-detector type, but it offers the advantages of lower acquisition cost and SPECT imaging capability. Multi-wire Proportional Chamber (MWPC) machines are a recent development which employs ionizing gasses and three wire planes to localize the γ ray/gas interaction. The achievable resolution with this technology is very good, about 4mm, but it presently suffers from its difficulty in stopping the photon in the gas, only 1% of the γs which pass through the gas interact. This class of machine is still very much in the research phase of its development. Multi-detector systems can be constructed using hexagonal arrays of large crystals; parallel, opposed arrays of small crystals in a ring shape, or multiple rings of small crystals. The multi-detector system is in wider use than either the camera or MWPC machines, and it offers higher resolution and better image statistics for a given scan duration. For these reasons, the operation of the multi-detector machine is described, with some emphasis placed on the multi-ring machines.

A block diagram of a multi-detector PET system is presented in Figure 2.10. The system consists of one or more rings of gamma ray detectors mounted on a gantry, electronics for coincidence detection and γ ray origin computation, image processing equipment, CRT, patient table, and data archive. The γ ray detectors in common use are composed of bismuth germanate (BGO) crystals backed by a single photomultiplier (PM)

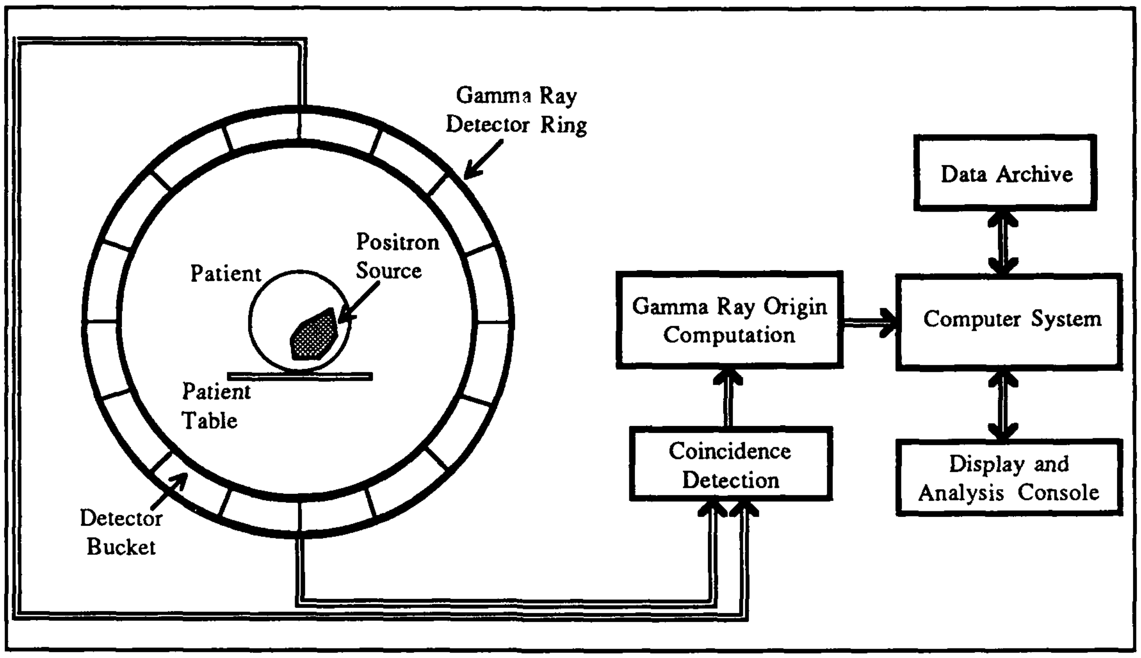


Figure 2.10: Diagram of a Positron Emission Tomography Imaging System.

tube. For ease of maintenance, the crystals and their PM tubes are grouped into arrangements called buckets, which can be remo'ed and installed as a unit. BGO crystals are used rather than NaI because they resist scattered photons better, stop the 511 keV γ rays better, and emit light in a shorter pulse. These advantages improve the count accuracy and increase the number of counts which can be gathered in a time interval. The drawback to BGO is that the light intensity is very low, forcing the use of PM tubes which can amplify low level light, thereby driving the cost of the PM tubes up.

The data for the image is gathered using the detector ring to record annihilation events. Because the detector is placed completely around the patient, PET can achieve attenuation correction and provide both sensitivity and resolution unattainable by SPECT. The first step in the data collection process occurs when the radionuclide in the patient's body undergoes positron producing radioactive decay. The photons produced by the annihilation event pass through the body and strike the face of the opposing gamma ray detectors. When a crystal interacts with a photon, it releases scintillation light, which is amplified by the PM tubes. The magnitude of the signal generated by the PM tube is directly proportional to the energy of the photon. A lower limit energy level of 200 keV is commonly used to filter out the photons which were scattered within the body before striking the detector. If the photon has sufficient energy, the signal indicating a photon/crystal interaction is sent to the coincidence electronics. The coincidence electronics are designed to accept these events and to determine if a coincident event occurred on the opposite side of the gantry. A timing window of 24 ns is used to establish coincidence, those γ photon pairs which strike detectors separated by $\approx 180^\circ$ within the timing limits imposed by the window are accepted as arising from the same annihilation event. The location of each of the interactions determines the line of response (LOR), which is a line connecting the two coincident detectors. Single events and triple coincidences are also kept track of for later use in correcting the image for false coincidences. A real-time sorter takes the LOR events and sorts them for use in image reconstruction. When the event has been placed into the correct LOR, the accumulator for that LOR is incremented by one. If there are multiple rings of detectors, cross-plane coincidences are also detected, so that for an n-ring detector, 2n-1 planes can be reconstructed. To increase the linear sampling, and thereby increase spatial resolution, some systems wobble the detector ring during image acquisition. To image an additional set of planes, the patient table is repositioned and additional counts are gathered.

Upon completion of the data gathering stage, a set of transverse slices of the imaged volume is constructed. Image reconstruction can be accomplished using either iterative algebraic methods or analytical methods such as filtered back-projection. Until recently, the analytical methods were the accepted reconstruction method, but the iterative methods have been receiving more interest because they can capitalize on prior information embedded in the data and the statistical nature of the measurements themselves. However, commercial systems employ the filtered back-projection method.

Image quality is determined by several, interrelated, factors. These factors are the choice of molecule to be labeled, the false coincidence rejection capability, the sensitivity and size of the scintillation crystals, the length of the timing window, crosstalk between detector crystals, and the image reconstruction algorithm. The molecule choice influences image quality because the amount of radioactive isotope expended before injection increases as molecule fabrication time increases. The time factor is critical to PET imaging because, unlike SPECT, the radioisotopes have short half-lives, thereby limiting the amount of time the isotope can spend in the molecule fabrication process. As a result, the molecules which can be used are limited to those which can be fabricated in 2-3 isotope half-lives.

PET, unlike SPECT, allows an increase in both resolution and sensitivity through crystal dimension manipulation. The resolution of the system increases as the width of the crystal decreases, as system resolution is defined to be 1/2 the crystal width. Crystal sensitivity increases as crystal depth increases. Since narrower crystals allow fewer LOR between opposing crystals, the accuracy of the coordinates assigned to the annihilation event improves as crystal width decreases. However, narrow crystals increase detector crosstalk due to γ rays passing through a crystal to its neighboring crystals or by Compton scattering. Crosstalk can be reduced by placing heavy metal shields around the crystals, which also improves the statistics for the system, but the shielding increases the distance between crystals, thereby degrading resolution. Deeper crystals, or denser crystals, stop more gamma photons, which increases system sensitivity since sensitivity is defined as the number of detected photons relative to the number which reach the detector. With this last issue in mind, alternatives to BGO, such as barium fluoride and cesium fluoride, have been developed because their higher atomic number serves to stop more γ rays. These crystals have the added benefit of having a rapidly formed, very short duration light pulse produced for each crystal/photon interaction, thereby improving temporal resolution. These crystals, however, also have their drawbacks. The most severe are the low light yield they possess and the requirement that they be protected from moisture. The low light output increases system cost, the protection requirement prevents close packing of the crystals, thereby decreasing system resolution.

The duration of the timing window is largely determined by the response time of the scintillation crystals used. For crystals with a long response time, the timing window must be increased so that random coincidences are rejected, with the result that resolution is degraded due to noise. For shorter response times, the window can be narrowed, improving the chances that random coincidences are rejected and so the noise in the image is decreased. For crystals such as barium fluoride and cesium fluoride, the response times are short enough so that time-of-flight (TOF) calculations can be made in addition to using a narrow timing window. In this mode of operation, the difference in time between the occurrence of the crystal/photon interactions deemed to mark one annihilation is computed. Given this information, and the distance between the two crystals that participated in the event, the approximate position of the annihilation event can be computed. This knowledge can then be used during back-projection to improve the image since there is a smaller spatial distribution of probable locations of each event that comprise each LOR. Only those elements of the 2D matrix which fall within the probable location of the events on the LOR are allocated a weighted portion of the LOR accumulator total. An important additional effect of TOF PET is that the event acceptance rate for each crystal can be increased, thereby improving the image statistics, and random coincidence detection is improved. The final image benefits from these advantages in that it is a more accurate reconstruction because of reduced noise, better 3D resolution, and improved statistics. The cost comes in the form of additional PM tubes, one for each crystal, and more expensive electronics in an already expensive machine.

With these image quality considerations in mind, a modern PET scanner can be characterized as one employing multiple rings of detectors using a large number of narrow barium fluoride or cesium fluoride crystals, with relative TOF techniques used to enhance image resolution. Transverse slice thickness is limited to 8mm, and system resolution is between 4 and 6mm, which is close to the limiting theoretical resolution of 2mm. The system exhibits little noise, and good statistics due to the number of photons counted.

2.7 Conclusion.

In the preceding pages various approaches to the problem of providing diagnostically relevant medical imaging information were presented. The performance and operation of these modalities is summarized in Table 2.3. Unless otherwise indicated, figures are from references in the text.

These techniques rely heavily on the use of image reconstruction techniques to form images based on the body's response to different forms of energy. While the interpretation of the data, and the physical processes it portrays, varies greatly from modality to modality, the underlying theme of all approaches is the effort to provide the physician with information that aids in the formation of a diagnosis.

As each modality provides different information, there is no one "best" technique, but rather the clinician has an armentarium to choose from, and can select the technique which best addresses the problem at hand with a minimum of patient risk. Even though each technique uses a different physical property to form the image, all are alike in that a representation of the 3D volume that was analyzed can be built up from the sets of parallel, equally spaced 2D slices that each constructs. For the purposes of image rendering, the origin of the image data is not a concern as long as it is presented in this 2D slice format. The next chapter describes how the image data supplied by these medical imaging modalities can be modeled and displayed graphically in three dimensions.

Table 2.3
Summary of the Operation and Imaging Parameters for the Five Medical Imaging Modalities

Parameter	MODALITY				
	CT	MRI	Ultrasound	SPECT*	PET#
Interslice distance	1 - 15mm[1]	2 - 10mm[2] 3 - 20mm[3]	1 - 3mm[4]	9 - 15mm	5 - 9mm[5] or 4.9 - 8.1mm[6]
Reconstruction Matrix Size	256 x 256 or 512 x 512[1]	256 x 256[7] or 512 x 512[8,9]	128 x 128[3]	64 x 64 or 128 x 128[10,11]	256 x 256 up to 1024 x 1024[6]
Resolution	.5 - 2mm or .7 - 2mm[12]	1 - 3mm[6] or .5 - 2mm[13]	1.27mm[4]	7 - 10mm[14] or 3.5 - 9mm[10]	4 - 6mm or 2.3 - 6.6mm[15] or 5.8 - 7.8mm[6]
Imaging Mechanism	Transmission X-rays	High frequency RF transmission and external magnetic fields	High frequency sound	Emission γ rays	Coincidence timing of γ ray pair produced by β^+ annihilation
Physical Measurement	X-ray linear attenuation	RF pulse stimulated emission of an RF signal from in-phase precessing protons	Acoustical impedance	Uptake of radioactive nuclide labeled biochemical compounds	Uptake of biochemical compounds made from radioactive isotopes
Interpretation	Material density	Mobile hydrogen density contrast enhanced by T_1 and T_2 weightings.	Material boundary and homogeneity	Biochemical activity	Biochemical activity

* Resolution given for FWHM.

# Resolution given for FWHM, non-wobble mode.

[1][Pic89] and [Ful88]. [2][Spr87], [Mor86], and [Won87]. [3][GE88] [4][Gre82]
[5][Der86]. [6][Sca89]. [7][Mor86]. [8][Kan88]. [9][Pat89b] [10][Pat89a]
[11][Tos88], [Tos89]. [12][Ful88] and [Spr87]. [13][Kan88]. [14][Spr87]. [15][Kar88].

CHAPTER III

DATA MODELS AND GRAPHICS OPERATIONS

3.1 Introduction.

Before examining medical image processing machines, it is appropriate to first investigate the techniques used to manipulate medical image data with a view toward understanding their impact on medical imaging machine design. This chapter opens with an examination of the data models used in medical imaging applications. This discussion is followed by a definition of the coordinate systems used in graphics processing operations. The chapter concludes with a description of the graphics operations employed in medical imaging. Detailed descriptions of the graphics processing operations discussed can be found in a standard graphics text, such as [Fol83] and [Rog85]. However, before beginning, a short description of the assumptions employed and the rationale for presenting this material is provided below.

Medical imaging uses the following five classes of graphics operations to place a realistic three-dimensional (3D) image on a two-dimensional (2D) screen: geometric transformation, surface recognition, hidden-surface removal, anti-aliasing, and shading. The following discourse on these five graphics operations focuses on their operation, and assumes that the algorithms operate on the primitive data elements (either voxels, cuberilles or octree leaves, defined below) found in the volume approach to medical imaging. Whenever an algorithm can be applied under both the octree and cuberille data models, the discussion is presented in terms of primitive data elements (pdes). There are substantial differences between the algorithms for hidden-surface removal and geometric transtormation under the cuberille and octree data models, so for the sake of clarity our analysis for these two classes of algorithms is limited to algorithms which are used with the cuberille data model. However, to be complete, algorithms for hidden-surface removal and geometric transformation under the octree model are presented in Appendix B.

The timing analyses presented for algorithms in this chapter are based on common algorithms in each class and assume an implementation on a serial computer. The purpose of the analysis is to impart an understanding of the computational costs encountered when forming high-resolution simulated 3D images. The analysis does not attempt to identify the most efficient algorithm in each class or to examine the performance of these algorithms in a parallel processing machine. Note, too, that the cost of increasing 3D image resolution is high because the number of primitive data elements to be manipulated increases rapidly as resolution increases. However, due to the nature of the medical imaging environment, high resolution is required, and such increased resolution can be achieved only by increasing the number of primitive data elements. Therefore, operations on the primitive data elements must contend with the ever increasing run time associated with increased resolution while still providing reasonable speed. The requirement for high resolution, and the associated large computation cost, drive the design decisions made in the machines examined in Chapter IV.

3.2 Data Models.

This section opens with a discussion of the voxel data model. The importance of this model derives from its widespread use. The voxel model is the basic representation used in medical imaging modalities and is the assumed raw data format for the three major approaches to medical image data modeling. The second section describes the three major approaches which have been developed for data modeling in medical imaging: the contour approach, the surface-tracking approach, and the volume approach. Then two models specific to the volume approach, the cuberille model and the octree model, are described.

3.2.1 The Voxel Model.

Voxels are identically sized, tightly packed parallelepipeds. The voxels must be non-overlapping and small compared to the features to be represented within a volume. The lower limit on the cross-section of a voxel is the resolution of the modality which gathered the image data. Representative values for voxel dimensions in a CT slice is .8mm x .8mm x 1mm. A voxel representation of a cube-shaped space is presented in Figure 3.1.

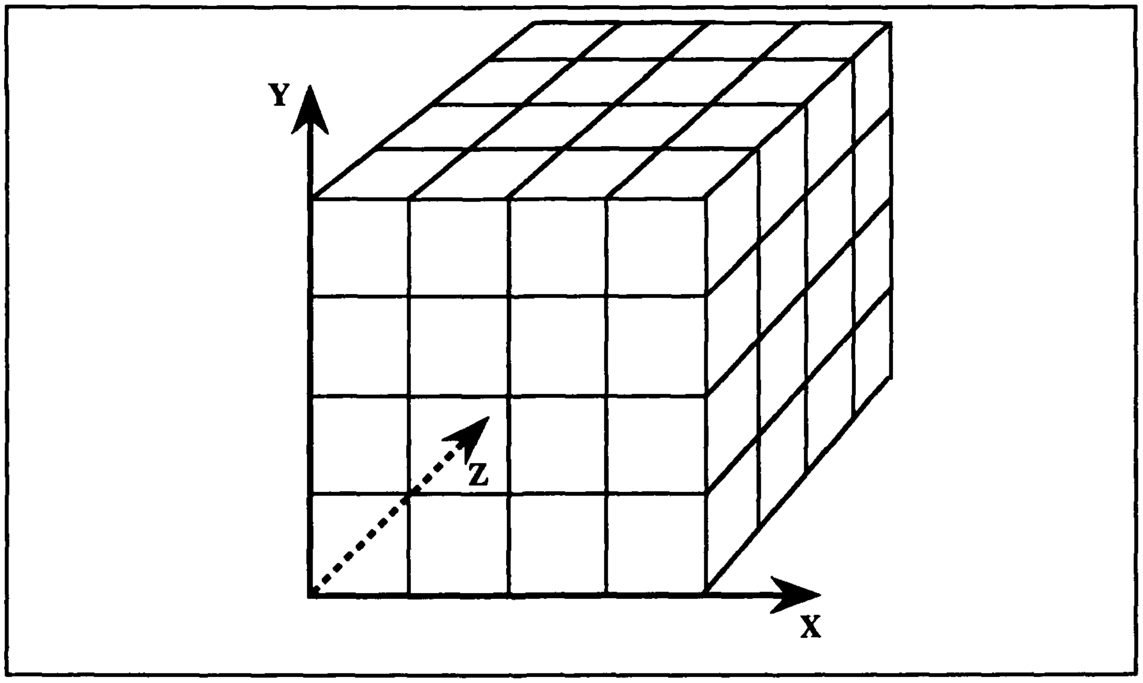


Figure 3.1: Voxel Representation of a Cube in Object Space.

A simplified representation of a CT slice using voxels is presented in Figure 3.2 (from [Rob85]).

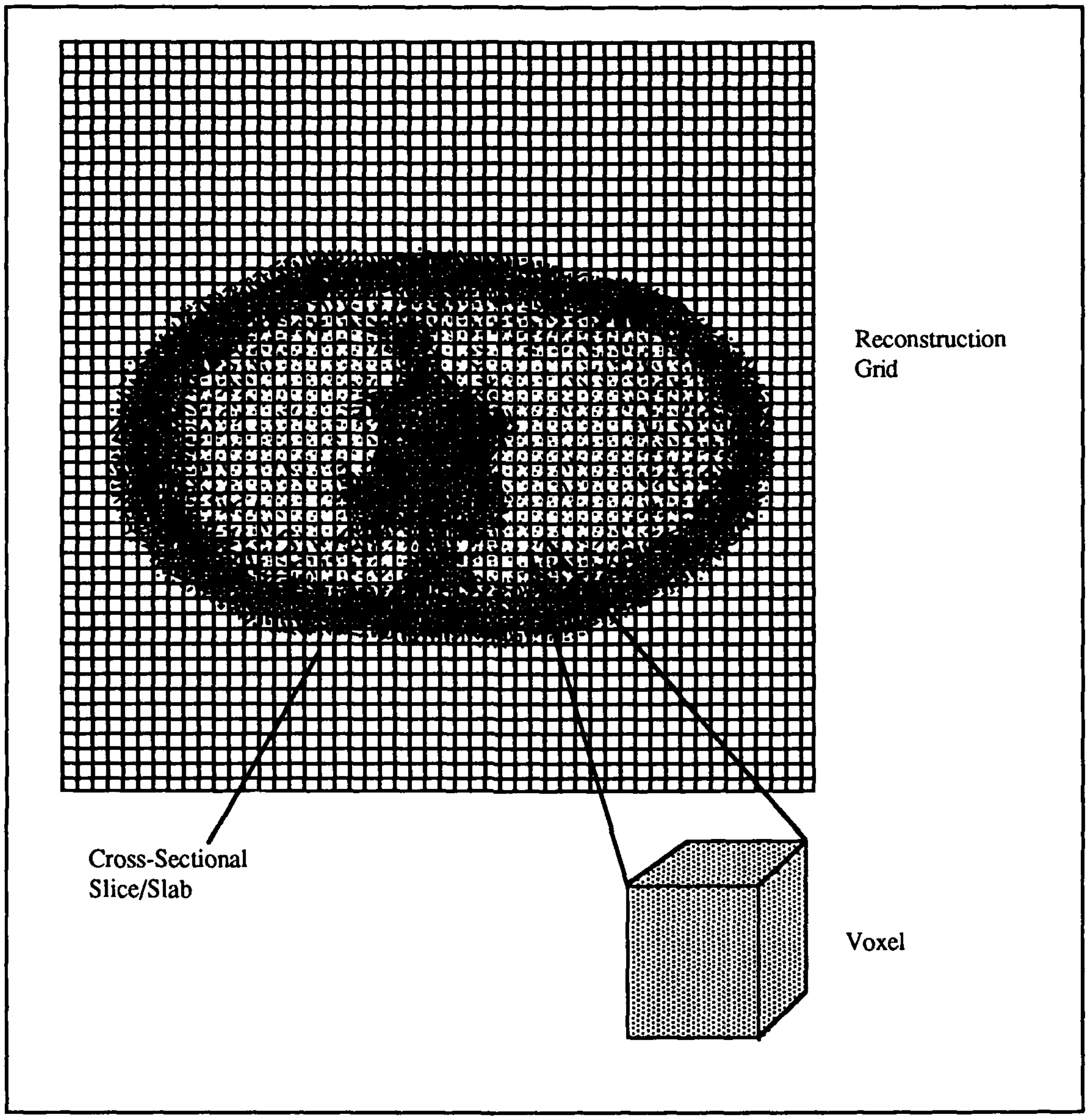


Figure 3.2: Voxel Representation of a CT Slice.

The voxel array is naturally represented by a 3D array, in which each array element corresponds to a voxel in the 3D digital scene. The value of an array element is the CT number of the corresponding voxel.

3.2.2 Medical Imaging Data Models.

To accommodate the image representation requirements imposed by a 3D medical imaging environment, three major approaches, contour, surface, and volume modeling, have been developed. The techniques are briefly summarized here, [Far89], [Gol86], and [Sri81] provide thorough surveys of these approaches. Each approach is based upon a different data primitive, which is either one-dimensional (1D), 2D, or 3D unit-based. The dimension of the unit is determined by the dimensionality of the primitive data element used in the model. For example, a data model which uses lines to construct images is a one-dimensional model, a model which uses squares to construct images is two-dimensional, and so on. All three approaches assume a 3D array of input data derived from a CT, MRI, ultrasound, PET or SPECT study.

Contouring and surface modeling attempt to provide rapid image generation through the use of models of the edges of an organ, thereby reducing the amount of data which must be manipulated to form an image. These techniques suffer from the requirement for re-processing the volume to extract other organs or to change the viewing parameters. Volume rendering techniques avoid the image re-processing but pay a computational penalty due to the volume of data they must manipulate to generate an image. A discussion of machines which perform medical imaging based on contour and surface approaches is presented in [Gol88].

The contour approach uses one-dimensional (1D) unit-based models to depict the surface of an object within a voxel-based set of slices of a volume. Each set of 1D contours provides a piecewise linear description of the boundary of the object of interest within each individual slice of a 3D volume. Objects are modeled based on the intersection of the object surface with each slice; voxels at the intersection of surface with slice form the border of the object in a particular slice. The contours can be isolated by thresholding if the boundary occurs between two high-contrast materials or by using automatic or manual tracking. Due to the complexity found in portions of the typical medical image, the automatic methods generally require operator assistance to accurately extract the boundary of an object. Contours are economically represented by chain code and quadtree data structures.

The surface approaches to 3D visualization form the second data-modeling category. These models are based on 2D primitives and commonly employ triangles or polygons to depict the surface of a selected portion of the volume. Because the data from a CT, MRI, PET, SPECT, or ultrasound scanner has gaps between the slices, this approach requires inter-slice interpolation to approximate the inter-slice object surface. There are two sub-categories in this class, tiling and surface tracking.

Tiling techniques (see [Fuc77] for an example) require extraction of the contours of the object of interest in each slice using any of the methods employed for the contour approach. Tiling models base the tiles on the extracted contours of the individual slices. The extracted contours are smoothed and then geometric primitives, typically triangles, are fitted to the contours to approximate the surface of the object in the inter-slice space. Tiled surfaces furnish realistic representations of the objects selected for display, especially when objects are differentiated with color and multiple light sources are used to light the scene. Because of the data reduction inherent in going from the input data to a tiled surface representation, the object can be rotated quite rapidly. The disadvantage inherent in this approach are the time required to extract the surface of the object and the requirement for re-accomplishing object contouring if a different object is selected or if an interior view of the object is desired.

2D surface-tracking methods (see [Art79a], [Art81], [Her78], [Her79], [Liu77], [Udu82a], and [Udu82b]) define a surface in a binary scene as a set of connected polygon faces which separate polygons of value one from polygons of value zero. These techniques begin by interpolating the original voxel-based image data to form cuberilles (defined in section 3.2.2), yielding a volume representation composed of cubes. The structure to be displayed is then isolated by applying a threshold operator to the volume to form a new binary representation of the 3D volume. Binary cuberilles located within the object are assigned a value of one, all other cuberilles are set to zero. Surface tracking is initiated after a seed point in each component of the object has been identified by the user. The surface-tracking algorithm then processes the volume and selects the cuberille faces oriented toward the observer for display. The extracted object border is represented by a set of face descriptors. The descriptor contains the location of the border voxel, the direction of the face, and the relative position of adjacent faces. The direction the face points toward and the relative orientation of adjacent faces is used to estimate the local surface normal for image shading operations.

The 3D unit-based volume approach has grown in popularity due to its ability to render images without a preprocessing step as the computational power which can be applied to the image rendering task has increased. This category contains the voxel, octree, cuberille, and other parallelpiped-based models. This class of models portrays a scene using volumes and so eliminates the need for performing surface recognition or for forming inter-slice tiles. The volume techniques use the 3D primitive generated by the imaging modalities, the *voxel*, as the basic data type and so require the least preprocessing of the three approaches. The drawback to this approach is the volume of data which must be processed each time a new view of the volume is constructed. The imaged volume is represented within the machine as a 3D array, with access based upon row, column, and depth coordinates. By using back-to-front access to the matrix, as described in [Fri85a], the visible portion of the objects within the volume can be rendered to the display. Because of the computational cost involved in generating an image this way, variations on this theme have been developed which attempt to reduce the number of data elements processed. For examples see [Gol87], [Rey85], and [Rey87]. The octree and cuberille data models are discussed in detail below.

3.2.3 The Cuberille and Octree Data Models.

This section presents a description of two data models which are used in the medical imaging field, the octree model and the voxel model. The discussion concentrates on the basic definitions and data elements used by each model and is based on the information found in [Her82]. Both the octree and cuberille model representations of a volume are built up from the voxel representation for the volume produced by the CT scanner.

3.2.3.1 The Octree Model.

The octree model is an extension of earlier work on *quadtrees* (see [Hun79]). The twofold goals of the model are to support modeling of complex surfaces and to minimize the amount of computation required at runtime to form an image. The model can be used to portray three-dimensional objects at any resolution and has the further advantage of being able to be implemented on inexpensive processors. The octree model minimizes runtime computation costs by performing a preprocessing step which identifies continuous volumes in a 3D scene which are identical in some property, such as CT number or T_1 relaxation time. These homogeneous volumes are identified by examining all of the voxels in the volume, and representing homogeneous connected volumes as leaves on an eight-ary tree (octree). Branches in the tree signify volumes in the 3D space which are not homogeneous. The octree is built up from terminal leaf nodes by using the branches to represent volumes which contain leaves and/or other branches. The tree thus formed represents the entire space that was imaged. These branch and leaf nodes in the tree, rather than the underlying voxel data, are then manipulated as units. An excellent description of the octree model can be found in Meagher's 1982 paper [Mea82a]. The description and explanations provided below are based on this work and work presented in [Doc81], [Hard84], [Jac80], [Kun85], and [Mea82b]. The next paragraphs present the basic definitions used in the octree model as well as the procedure to follow to build an octree.

The universe that the octree model depicts is a section of three-dimensional space defined by three orthogonal axes. The displacement in a dimension is bounded by $0<x(i)<e$, where $x(i)$ is the displacement in dimension i and e is the length of dimension i in the universe. A given object B is defined as a family of ordered pairs, $B(k) = (P,E(k))$. P is a finite set of properties and $E(k)$ is the set of disjoint primitive data elements which exactly fill the universe at resolution k. For a binary image, property P describes the state of each octree node, the three permissible states being Full, Partial, and Empty. In the context of medical imaging, the assignment of the value Full correlates to a volume falling within a user-defined CT number window; conversely the Empty assignment indicates that the associated volume falls outside the window. The Partial assignment indicates that the corresponding volume has primitive data elements which fall into both the Full and Empty categories. Octrees which provide gray-scale images can also be built, but the depth of the octree and the memory required to hold the octree increases accordingly.

The basic approach to building a binary octree is to recursively divide a given space until all terminal nodes are labeled either empty or full. The cubic regions formed by this recursive subdivision are called octants. A non-terminal octant is assigned a value of partial, and is a branch of the tree. The octree representation of a binary volume can be formed as follows:

> Examine the densities in a volume. If they are all one, represent the volume in the octree by a terminal node labeled Full. If they are all zero, represent the volume by a terminal node labeled Empty. Otherwise, label the node Partial and create eight children of the current octree node. Assign to each child a distinct volume formed by subdividing the volume of the parent into eight sub-volumes. Each of the sub-volumes may in turn be represented by octree nodes labeled Full, Empty, or Partial. Subdivision ceases when all terminal nodes are labeled Full or Empty.

When the above algorithm is applied to the cube in Figure 3.3, the octree shown in that figure is generated.

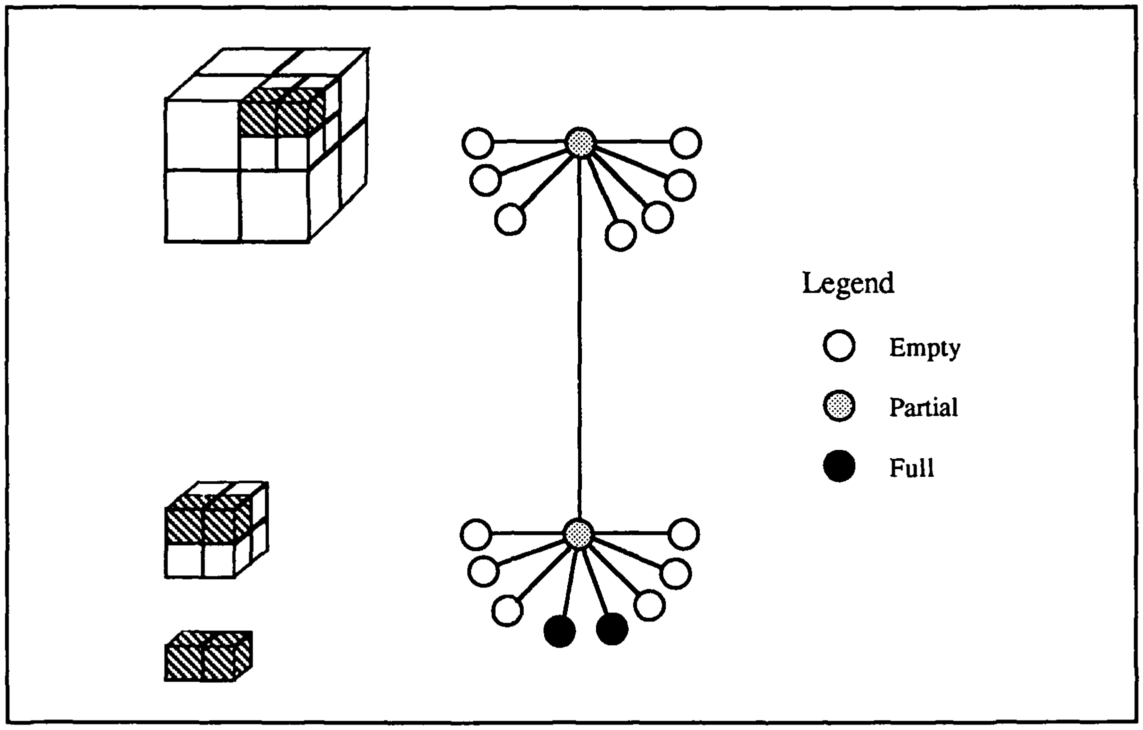


Figure 3.3: Generating an Octree.

<u>3.2.3.2 The Cuberille Model.</u>

The cuberille model has been used more extensively in medical imaging than the octree model. The following discussion, based upon the foundation paper for this model, [Her79], addresses the characteristics of the cuberille model, present its basic definitions, and compares and contrasts it with the octree model.

The cuberille model, as described in [Her79], forms cuberilles from the voxel output from the CT scanner. A cuberille depiction of a scene,S, is defined as a pair [ϕ, C] where:

1. ϕ is an infinite collection of closed cubes which cover the whole of 3D space and are placed such that if two components of ϕ intersect, then the intersection is a face of both components.

2. C is a mapping which associates with every face of S the point of the center of the face.

As mentioned earlier, cuberilles must be non-overlapping and small compared to the features to be represented. A representation of an object volume using the cuberille model can be implemented quite naturally with a 3D array, wherein each element of the array is assigned a value based upon the CT number of the corresponding cuberille(s). Interpolation of voxel values in order to determine cuberille values is often required.

The most immediately apparent difference between the cuberille and octree models is in their subdivision of space. The octree model seeks to subdivide space with no regard for the size of each elementary element. It is necessarily true that the nodes at the same level of an octree represent equal volumes of object space. However, since leaves and branches are intermingled on the same level, the leaf nodes can vary greatly in the amount of object space volume they represent. This follows from the basic octree building criterion that each terminai node be uniform in some physical property such as CT number, not in the size of the object space volume represented.

On the other hand, the cuberille model uses identically sized cubes of space to model the object. In this case, the model requires that object space be equally subdivided with no regard for the homogeneity of the volumes thus formed. The value of the cuberille is a characteristic of a physical parameter over the volume that the cuberille represents. A useful way to differentiate the two models is as follows. The cuberille model employs the image data produced by the imaging modality, possibly with some interpolation. The octree model takes the CT scanner data, and in a preprocessing step, forms a higher level model of the object space. The advantage of the cuberille model is that it requires less preprocessing time than the octree model and so permits the initial view to presented quite rapidly. On the other hand, the octree model requires less computation for the second and subsequent images formed of the 3D space than the cuberille model.

3.3 Coordinate Systems.

Throughout the remainder of this dissertation, I employ two different coordinate systems, the object-space coordinate system and the image-space coordinate system. The coordinate system in which the pdes (and therefore the objects in the scene) are defined is referred to as object space, and the coordinate system in which the pdes are displayed as pixels on a CRT is the image space. Figure 3.4 portrays the relationship between these two spaces, with the direction of positive rotation about each axis as indicated.

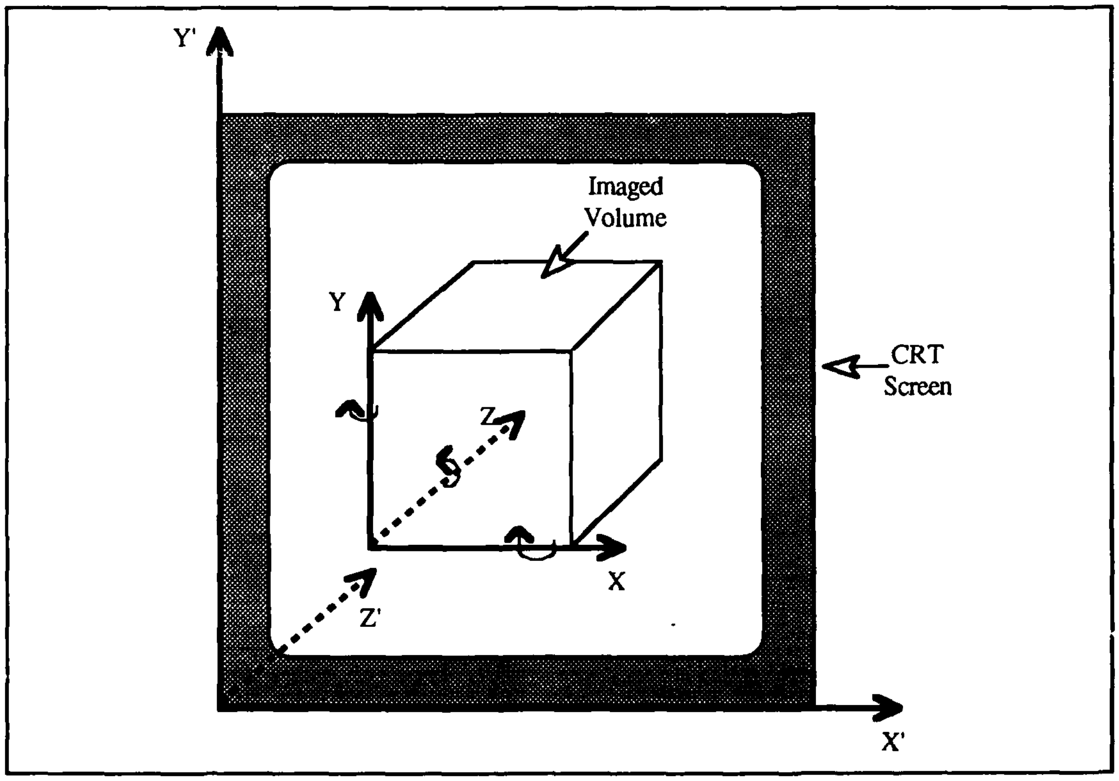


Figure 3.4: Object and Image Space Coordinate Systems.

The two coordinate systems are related to each other through the coordinate system transformation operations described below. The object-space coordinate system, signified by X, Y, and Z, is fixed in space. We fix these axes for three reasons. First, the transformations are always applied to coordinates located at their original, unrotated position; thereby requiring that the translation values used when rotating the scene be computed only once. Second, the matrices can be computed so that round-off errors do not accumulate, which is especially important when applying the rotational matrices. Third, the additional computation required to maintain orthogonal matrices and to eliminate object distortions arising from roundoff errors in previous transformations is eliminated. The image-space coordinate system, signified by X', Y', and Z', is constrained to rotate along with the representation of the scene as the coordinate transformation matrices are applied. The object-space object representation remains unchanged as a result of the transformations, and thus the changes to the representation occur only in the image space.

3.4 Geometric Transformations.

The three geometric operations of rotation, scaling, and translation are quite similar in terms of processing requirements: each primitive *data element (pde) must* be manipulated in order to effect the transformation. When rotating and translating, each pde is examined once to determine the CT number for that volume, and then the CT number is mapped into a new location based on the amount of translation or rotation to be performed. When scaling, each pde must again be examined once, but in this case, the average of the CT numbers over a volume is mapped into a new location. The following paragraphs contain a description of the underlying mathematical operations that must be performed in order to accomplish these transformations under the cuberille model. In this discussion, [x y z 1] represents the original coordinate of a cuberille in the scene and [x' y' z' 1] the transformed coordinate.

Translation is the process of moving objects in xyz space to new positions within the same space by adding a user-specified translation amount to the coordinates of each point of an object. If D_x, D_y, and D_z represent the displacement along the respective x, y, and z axes, then the equation for the translation of a single cuberille can be written as:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ D_x & D_y & D_z & 1 \end{bmatrix} \qquad (3.1)$$

The above equation is applied to each cuberille in the scene, and then the result of the translation is displayed. The effect of successive translations is additive.

Scaling alters the apparent size of objects within the scene by multiplying the endpoints of vectors representing the edges of the objects in the scene by the appropriate scaling factor for each dimension. If S_x, S_y, and S_z are used to represent the scaling factors along the x, y, and z dimensions, then the scaling operation for a single cuberille can be expressed as follows:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1] \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (3.2)$$

Scaling the entire scene requires that each vector endpoint in the scene be scaled. Successive scalings are multiplicative in effect on the size of objects in the scene, i.e., a scalin^r f .5 on a scene followed by another scaling by .5 shrinks all the objects in the scene by 75%.

Rotation about the origin is simply expressed as a set of rotations about each of the coordinate system axes. Successive rotations are cumulative in effect on the angular position of the objects in the scene relative to their starting positions. For example, in a scene with a single object, a rotation by 45° about the x-axis followed by a rotation of 60° about the x-axis results in a 105° rotation of the object about the x-axis. If the angular rotation about the x-axis by is represented by θ, the following equation for a rotation of a point within an object about the x-axis results:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

For a rotation of β degrees about the y-axis:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1]\begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

And a rotation of γ degrees about the z-axis is expressed by:

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1]\begin{bmatrix} \cos\gamma & \sin\gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

Rotation of the scene requires that each of the above equations be applied to each cuberille in each object before it is displayed. Rotation about an arbitrary point is a more involved process than that described above. First, calculate the amount of translation along each axis that is required to displace the point the scene is to be rotated about to the origin of the object space coordinate system; the translation amount along each axis becomes one element of our displacement vector. Then, apply the displacement vector to each cuberille, rotate the cuberille to its new position using the rotation matrices described above, and then apply the negative displacement vector to the cuberille. After each cuberille has been processed the desired image can be displayed.

3.5 Image Segmentation.

Image segmentation seeks to differentiate the surface of the object(s) of interest from the remainder of the object space. This section begins with a discussion of image segmentation in general and then proceeds to an examination of two representative algorithms. Then boundary detection, a form of image segmentation which has found some popularity in the medical imaging community, is discussed.

3.5.1 Image Segmentation.

Image segmentation is a technique for locating those regions in a scene that have the same property. Performing image segmentation, therefore, requires examination of each cuberille in the 3D scene, resulting in computational complexity that grows linearly with the number of cuberilles. In general, there are two steps involved in the process. First, the cuberilles in the scene must be classified according to some user criteria, for example, a user-defined threshold which separates the object(s) of interest from the remainder (background) of the scene. A cuberille within the threshold is assumed to be part of the object, otherwise it is classified as background. Second, all the cuberilles in the scene that are in the same class and are connected must be located. The total number of cuberilles that are examined can be reduced by having the user indicate a seed cuberille within the object and then tracing the remainder of the surface of the object using the seed as a starting point. However, the running time still grows linearly with increases in 3D resolution.

As there is no general theory of image segmentation, the process is illustrated with two simple algorithms, a generic algorithm and Farrell's algorithm, see [Far84] and [Far85].

Assume a 2D scene composed of pixels in which there is one object. The scene size and the screen resolution are identical. Because the scene was formed by a CT scan, the pixels in the scene that are *w'thin the object of interest* must all lie *within a single range* of CT numbers. We wish to form a binary scene by segmenting the scene so that all the pixels on the screen *within the object* are given a value of one and all background pixels are given a value of zero. One way to accomplish this task is as follows. First, the user indicates the range of CT numbers which form an acceptable window for CT numbers for the object. Second, starting at the upper left-hand corner of the scene, determine the values for screen pixels as follows: if the scene pixel has a CT number within the CT number window, assign the corresponding screen pixel a value of one; if its CT number is outside the window, assign the pixel a value of zero. Perform this operation for all the scene pixels. When the process terminates, the object is displayed on the screen.

Farrell's technique (see [Far84] and [Far85]) is also simple. The object of interest is differentiated from the remainder of the scene volume by assigning a unique color to each range of CT numbers that specifies an object. This technique assumes that the object corresponds to a continuous range of CT numbers (ie., there are no discontinuities in the threshold window), and that no other objects are incorrectly assigned the same color. These two assumptions give this technique the worst image quality of the boundary detection procedures presented here, but it is also the fastest procedure.

The image segmentation process becomes more complicated when several objects are displayed, if a gray-scale image is to be formed, and when there is a 3D scene to be segmented. Boundary detection algorithms attempt to resolve these issues in a computationally efficient manner.

3.5.2 Boundary Detection.

Boundary detection requires that the surface of the organ(s) of interest in the scene be identified and that this information be used to separate the organ(s) from the remaining portion of the 3D digital scene. We discuss three algorithms that have been employed for boundary detection: the boundary detection algorithm proposed by Liu in [Liu77], the boundary detection algorithm specified in [Art81], and the "marching cubes" algorithm described by Lorensen and Cline in [Lor87]. For all three algorithms, computational complexity increases linearly as the 3D resolution increases.

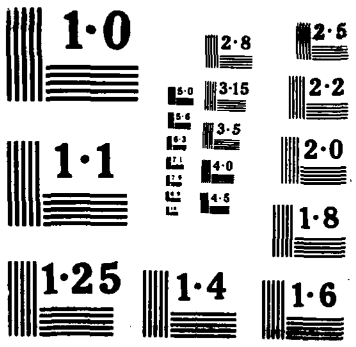
Liu's algorithm detects a 3D surface by exploiting three properties of an object : high contrast between the object and surrounding material(s), connectivity, and agreement with *a priori* knowledge. The connection of boundary elements is guided by these three properties, along with the requirement that at least *one* and at most *two* neighbors of every boundary element also lie on the boundary. The possible boundary elements are considered to be those lines in 3D space which have high gradient values. The initial boundary element is selected by the user and localized based on a maximum gradient criterion. The next, and all subsequent, boundary elements are found by examining the neighbors of the current boundary element and then choosing the element which has the highest gradient value and has not already been identified as being on the object boundary. The search for boundary elements is guided by a simple heuristic which tests the gradients found at each element against a minimum threshold value for the gradient. If it is determined that none of the gradient values at a given boundary element exceeds the threshold gradient value, then the algorithm backtracks to the immediately preceding boundary element. Backtracking continues until an acceptable gradient value is found (among those not previously used) at a boundary element. The search then resumes from the newly identified boundary element.

The second algorithm is the boundary detection algorithm described in [Art81]. Before describing the algorithm, a little background on the theory underlying the algorithm is required. This procedure for boundary detection models the surface of an object as a connected component in a directed graph (digraph), and translates the problem of surface detection into a problem of tree traversal. In [Art81] it is proven that, given a directed graph G whose nodes correspond to faces of voxels separating the object in the scene Q from background then the connected subgraphs of G correspond to surfaces of objects in the scene. Therefore, finding the boundary surface of an object in the scene corresponds to traversing a connected subgraph of a digraph. A key observation which makes the algorithm practical is that every node in the digraph has indegree two and ıtdegree two. This fact, combined with the fact that every connected component of the digraph is strongly connected, means that for every node there is a binary spanning tree rooted at that node which is a subgraph for the digraph and spans the connected component containing the given node. Therefore, given a single face of a cuberille on the boundary of an object, it is guaranteed that every cuberille face in the boundary of the object can be found.

The surface detection procedure works as follows. The algorithm assumes that the 3D scene is represented by a three dimensional array of voxels. The object(s) in the scene are defined to be a connected subset of the three dimensional array. The set of voxels that represents the object of interest is specified by setting an appropriate gray-scale value and then thresholding the scene. The user then specifies a seed voxel. This seed is used as the starting point for the algorithm. As each node in the binary spanning tree is visited, the node is made current and is checked against a list of previously visited nodes. If the node was visited previously, it is removed from further consideration as it can never be visited again. If the node has not been visited previously, it is added to the list of once visited nodes. The nodes connected to the current node are identified by locating the voxel faces

AD-A218 024 THREE-DIMENSIONAL MEDICAL IMAGE ANALYSIS USING LOCAL DYNAMIC ALGORITHM SE (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH M R STYTZ 1989 2/6

UNCLASSIFIED AFIT/CI/CIA-89-158 F/G 6/4 NL


1·0
2·8
2·5
5·0
3·15
2·2
5·6
3·5
6·3
2·0
1·1
7·1
4·0
8·0
4·5
1·8
1·25
1·4
1·6

which are adjacent to the current voxel face. The nodes connected to the current node are then added to the list of nodes to be considered. Then, the next node to visit is selected from the list of nodes to be considered based on a first-in, first-out discipline. The next node is then examined. The algorithm terminates when the queue of nodes to be considered is empty.

The third algorithm is the "marching cubes" algorithm outlined in [Lor87]. This technique processes the data in a medical image volume in scan-line order, creating a triangle model of a constant density surface as it moves along. The algorithm has two basic steps. First, the desired surface is located within the cube and defined by triangles. The second step calculates the normals to the surface for each vertex of each triangle in the cube. Surface location is performed by the marching cube, which is a logical cube created from eight pixels, four from each of two adjacent slices. The surface intersection(s) location within the cube is determined in a several step process. First, the data value at each vertex of the cube is examined, and a one assigned to a vertex if the value meets or exceeds the threshold criteria, otherwise the vertex is assigned a value of zero. Vertices with a value of one are within or on the boundary defined by the surface, those with a zero are outside the surface. This preliminary processing roughly locates the surface/cube intersection as the surface intersects the cube only along the edges where one vertex has a value of one and the other vertex has a value of zero. By inspection, the authors determined that there are 14 unique cases of surface/cube intersection to be considered, allowing determination of the type of surface intersection in the cube to be accomplished with a table look-up. Once the type of cube/surface intersection is determined, the location of the intersection along each edge is computed based upon a linear interpolation of the data values at the vertices of the edge. The point of intersection on each edge defines a vertex of a triangle. The type of cube/surface intersection is used to determine which vertices belong to a common triangle. For each triangle in the cube, the surface normal at each of its vertices is estimated by computing the gradient vector at each cube vertex and then taking the central difference of the cube vertex gradient values along each edge containing a triangle vertex. This completes processing for the cube, so the triangle vertices and vertex normals are output and the cube marches deeper into object space.

3.6 Anti-Aliasing.

Anti-aliasing is used to prevent jagged edges in the display and is accomplished by smoothing the edges of objects in the scene. Crow (see [Cro77b] and [Cro81]) presents three classes of anti-aliasing techniques. The three classes are: increasing the resolution; blurring or smoothing the contour plane; and making the sample point in the image correspond to a finite area in the scene. The last two algorithms are computationally intensive and so have not been widely used for medical imaging applications. Therefore, the discussion is restricted to resolution increasing techniques. Below, two resolution-increasing approaches to anti-aliasing are characterized, the Bartlett filter and super sampling to high resolution with simple averaging to low resolution. Both of these resolution-increasing techniques accomplish anti-aliasing by examining each pde in the scene. For both techniques, the run-time increases linearly with the number of pdes in the scene.

A description of the Bartlett filter algorithm is found in [Cro81]. To use this algorithm, each low-resolution pde is broken into m x m high-resolution pdes. The graphics processing operations are performed on the high-resolution pdes. The output intensity value assigned to a pixel is based on the low-resolution pde values which are computed by taking the weighted sum of the m x m high-resolution pdes lying within each low-resolution pde. The requirement for computing a weighted sum rather than an average adds an additional 25% to the computation time required for the algorithm.

Super sampling to high resolution with simple averaging to low resolution (super sampling) is conceptually simple. All that is required is that the graphics operations be performed at a higher resolution than the final display provides. For example, by calculating the image at a resolution of 1024 x 1024 and then displaying it at a resolution of 512 x 512 four pde values in the high-resolution image have been averaged into one pde in the final display. This algorithm escapes some of the time liabilities of the Bartlett filter because it uses a simple average of a 2 x 2 set of high-resolution pdes to form a low-resolution pde.

As noted above, each technique exhibits the same growth in running time with object resolution increase encountered in the algorithms examined previously. However, these two algorithms are not equivalent in their performance. Super sampling has a 25% lower running time than the Bartlett filter, but the images formed using super sampling are not of the same high quality as those produced by a Bartlett filter.

3.7 Shading.

The shading of a 3D object on a 2D screen is an important step in producing a depth cued image. In order to correctly shade a scene, a series of computations based on one or more of the following properties of each object being displayed must be performed: the refractive properties, the reflective properties (diffuse or specular), the distance from the observer, the angle of incidence of the light to the surface of the object, the surface normal of the object, the type of source of light (point or distributed), and the color of the object. Foley and VanDam [Fol83] and Rogers [Rog85] provide overviews of this material. Chen [Che84a] provides an excellent overview of several shading techniques used in the medical imaging environment.

This section opens with a brief discussion of the concepts employed when shading a scene. Then five shading algorithms are examined, distance shading, normal-based contextual shading, gradient shading, Gouraud shading, and Phong shading. The algorithms are presented in order of increasing computational cost.

3.7.1 Shading Concepts.

Shading is used to impart a photo-realistic, three-dimensional effect to rendered images through the use of an illumination model which characterizes the interaction of different types of light from different sources with the materials in the scene on a pixel-by-pixel basis according to the position of each material relative to an observer, position relative to each individual light source, and each material's composition and light reflection/transmission characteristics. Shading is a computationally intensive process which attempts to characterize the appearance of a rendered object using an illumination model to guide the sequence of computations. Illumination models are based on the concept that the amount of light reflected from and transmitted through an object makes the object visible. The wavelength of the incident light, in combination with the surface properties of the object, determine how much light is absorbed, reflected, and transmitted as well as the color of the object as seen by the human eye. The light reflected from the surface of an object can be characterized using the combination of wavelengths in the incident light, its direction, the type of light source (area, point, or diffuse), the orientation of the object surface, and the composition of the surface. As shading an object employing all of these parameters is computationally intensive, two classes of illumination models have been developed. Global illumination models attempt to characterize the appearance of each object by considering, for each object, the surface properties and orientation, the location, intensity, area, and color of all the light sources that shine upon the object, the amount of light reflected from surrounding objects, the distance from the light source(s) to the object, and the observer's position. These considerations lead to computations which determine the amount of refracted light transmitted through the object and the surface specular and diffuse reflection weighted by the distance from the light source to the object.

Local illumination models are simpler in that they base their computations on the surface orientation, illumination from a single light source along with an area light source, and the observer's position. A local illumination model need only compute the diffuse reflection at the surface weighted by the distance from the light source to the object.

The following global illumination model, from [Rog85], must consider diffuse and specular reflection as well as refraction effects at the surface of the object. Figure 3.5 contains a two-dimensional depiction of the relationships involved.

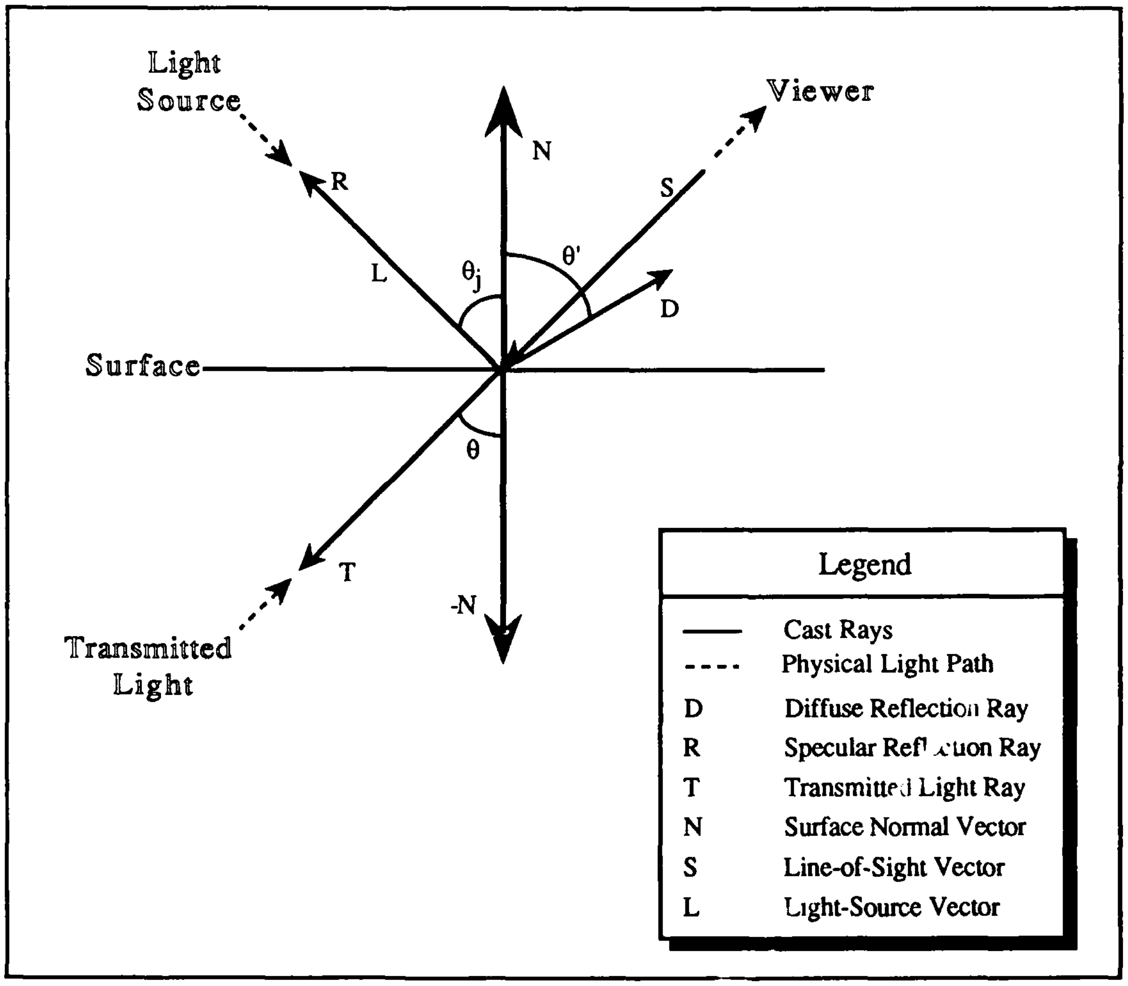


Figure 3.5: Surface Shading Relationships.

Diffuse reflection appears as a dull, matte surface and can be modeled using Lambert's cosine law for reflection from a point light source for a perfect diffuser:

$$I = I_l\, k_d \cos\theta \qquad 0 \le \theta \le \pi/2 \tag{3.6}$$

I is the reflected intensity, I_l is the incident light from the point light source, k_d is the diffuse reflection constant defined over the range 0 to 1, and θ is the angle between the surface normal and the light direction. Because of the large computational cost for computing the illumination for an area light source, area light sources are usually treated as a constant term and linearly combined with the diffuse reflection computed with (3.6):

$$I = I_a\, k_a + I_l\, k_d \cos\theta \qquad 0 \le \theta \le \pi/2 \tag{3.7}$$

The falloff in intensity for the point light source in (3.7) is linearly attenuated to yield:

$$I = I_a\, k_a + \frac{I_l\, k_d \cos\theta}{d + K} \qquad 0 \le \theta \le \pi/2 \tag{3.8}$$

where d is the distance from the light source to the object surface and K is an arbitrary constant.

Specular reflection gives rise to the shiny appearance on the surface of an object and is commonly modeled using an empirical formula described by Phong in [Pho75] or Cook and Torrance in [Coo81]. The following discussion is based on Phong's model which considers the angle of the incident light,*i* , its wavelength, λ, the angle between the reflected ray and the line of sight, α, and the spatial distribution of the incident light when computing the specular reflectance of an object:

$$I_s = I_l\ w(i,\lambda) \cos^n\alpha \tag{3.9}$$

The reflectance function, $w(i,\lambda)$, gives the ratio of specularly reflected light to incident light and $\cos^n$ is used to approximate the spatial distribution of the specularly reflected light.

Because w(i,λ) is a complex function, it is usually replaced by an experimentally determined constant k_s which is chosen to yield a pleasing appearance. Combining (3.8) and (3.9) yields the simple illumination model:

$$I = I_a k_a + \frac{I_l \ (k_d \cos\theta + k_s \cos^n\alpha)}{d + K} \qquad 0 \leq \theta \leq \pi/2 \qquad \textbf{(3.10)}$$

The model in (3.10) can be used to shade pixels on the screen by applying the equation once for each of the three primary colors. If multiple light sources are providing illumination, the effects of the light sources are linearly added, with the model becoming:

$$I = I_a k_a + \sum_{j=1}^{m} \frac{I_{lj} \ (k_d \cos\theta_j + k_s \cos_j^n \alpha_j)}{d + K} \qquad 0 \leq \theta \leq \pi/2 \qquad \textbf{(3.11)}$$

Common practice is to represent cosθ by the dot product of the unit surface normal vector $\hat{n}$ [1] with the unit vector for the light source direction, $\hat{\mathbf{L}}$, and to represent cosα with the dot product of the unit vector along the direction of the reflected ray, $\hat{\mathbf{R}}$, and the unit vector along the line of sight, $\hat{\mathbf{S}}$, making the model for a single light source:

$$I = I_a k_a + \frac{I_l \ (k_d (\hat{n} \cdot \hat{\mathbf{L}}) + k_s (\hat{\mathbf{R}} \cdot \hat{\mathbf{S}})^n \ )}{d + K} \qquad 0 \leq \theta \leq \pi/2 \qquad \textbf{(3.12)}$$

For a single light source oriented along the z-axis, $\hat{\mathbf{R}}$ can be computed using:

$$\hat{\mathbf{R}}_y = 2\hat{n}_z\hat{n}_y \quad \text{and} \quad \hat{\mathbf{R}}_x = 2\hat{n}_z\hat{n}_x \qquad \textbf{(3.13)}$$

[1]The surface normal can be computed directly from the plane equations which describe the objects in the scene or approximated by averaging the cross-products of the edges which terminate at each polygon vertex.

where $\hat{n}_z = \cos\theta$ (here θ is the angle between the unit normal vector and the z-axis). If the light source is not oriented along the z-axis, which occurs when using multiple light sources, the unit normal vector is translated and rotated until it lies along the z-axis. The direction for each reflected array can then be computed using:

$$\hat{R}_x = -\hat{L}_x \qquad \hat{R}_y = -\hat{L}_y \qquad \hat{R}_z = \hat{L}_z \tag{3.14}$$

More accurate models have been developed. Cook and Torrance [Coo81] developed a more comprehensive model by considering how material properties of each object reduce the intensity of reflected light. The model also includes computations to account for the blocking of ambient light by surrounding objects and the existence of multiple light sources of different intensities and different areas. In Section 3.8, a ray tracing based illumination model is described.

Reflection is not the only lighting effect to be considered. If transparent objects appear in the scene, then refraction effects must be considered. As incident and refracted rays lie within the same plane, the relationship between the incident and refracted angles is expressed using Snell's law:

$$\eta_1 \sin\theta = \eta_2 \sin\theta' \tag{3.15}$$

where η_1 and η_2 are the indices of refraction of the two mediums, θ is the incident angle, and θ' is the angle of refraction. To eliminate effects which can arise from an image space approach to refraction, the refraction computations can be computed in object space or in image space using a ray tracing model. Simple implementations of transparency effects ignore refraction and the decrease in light intensity with distance, and linearly combine the intensities from an opaque surface with the transparent surface lying between it and the observer:

$$I = tI_1 + (1 - t)\, I_2 \tag{3.16}$$

where I_1 and I_2 are the intensities computed at a transparent surface and opaque surface respectively and t is the transparency factor for I_1. If I_2 is transparent, the algorithm is applied recursively until an opaque surface is encountered or the edge of the volume is reached. This equation is identical to the compositing equation used by the Pixar and AT&T machines described in Chapter IV. More realistic transparency effects can be achieved by considering the surface normal to compute t.

Because of the computational cost incurred when employing a global illumination model, it is rarely employed for medical images, especially when rapid image generation is an important consideration. However, when combined with ray tracing, this type of model yields high quality images.

The algorithms in the next five sub-sections present several different approaches to rendering shaded surfaces. The first three models, constant, gradient, and normal-based contextual shading, are local models in that they do not consider lighting effects such as refraction in their calculations. They are complete as they stand and manage to provide usable medical image shading at a reasonable cost in time. The gradient and normal-based contextual shading algorithms are also notable for the simplifying assumptions they make for the computation of the local surface normal. The Gouraud and Phong models, on the other hand, are used to simplify the calculation of the intensity at points between polygon vertices. The Gouraud model relies on a global illumination model to provide the light intensity at each vertex and then uses this value to compute the intensity at points between vertices. Phong takes the normal computed at each vertex and interpolates these values along the edges in the scene, with the final intensity for each point determined using a global illumination model.

3.7.2 Constant Shading.

Constant shading is the simplest of the five shading techniques, since it merely calculates a single shading value for an entire polygon face. The technique can be applied with both point and/or ambient light sources. Constant shading assumes that the polygon faces represent the actual surface being modeled. This assumption causes the greatest difficulty in practice, because it makes each polygonal facet of the approximated surface visible, giving rise to the Mach band effect, which is the name given to the lines in an image that are produced along the intensity discontinuities which occur when there is a rapid change in the slope of the intensity curve in a portion of an image. The result of this assumption is that the intensity is exaggerated at any edge where there is a discontinuity in magnitude or change of degree of intensity. Constant shading can be mathematically modeled as follows:

$$I = I_a k_a + I_p k_d \left(\frac{\bar{L} \cdot \bar{N}}{r+k} \right) \tag{3.17}$$

where r is the distance from the perspective viewpoint to the surface, k, k_a, and k_d are user defined constants, $\bar{L}$ is the direction to the point light source from the surface, $\bar{N}$ is the surface normal, I_a is the intensity of the ambient light, I_p is the intensity of the point light source, and I is the intensity of light at the surface of the polygon. The run time of this algorithm increases linearly with 3D resolution increases.

3.7.3 Gradient shading.

Gradient shading was developed by the Medical Imaging Processing Group at the University of Pennsylvania in order to be able to provide a realistic shaded image at relatively low computational cost. Gradient shading, described in [Gor83b], [Gor85], and [Rey85], is different from the other algorithms discussed in this section in one important aspect. This algorithm is an image space, instead of an object space, shading algorithm, therefore the run-time of the algorithm increases linearly with the increase in 2D resolution. Since this technique also produces high quality images, it is well suited to the medical imaging environment.

The key to gradient shading is its use of the z gradient, which is computed at run-time and approximates the amount of change in the z-dimension between neighboring entries in the z-buffer. For best results, Gordon and Reynolds recommend that a weighted average of the forward and backward differences be used to compute the z-gradient. Once the gradient is determined, the local normal can be computed, and using this, the light intensity at the corresponding pde surface can be calculated.

The gradient shading algorithm operates as follows (from [Gor85]):

> Assume that at the end of the hidden surface removal process we have a z-buffer containing, for each pixel, the distance from the light source to the closest point on the object which projects onto that pixel. The z-buffer then contains a complete representation of the visible surface of the object of the form $z=z(x,y)$. We can compute the normal N at any point by finding the gradient vector ∇z at that point. Assuming the direction of light L is known and the other parameters can be estimated, the appropriate intensity can be estimated by applying $I = I_{max} * N \cdot L$.

Gordon and Reynolds report that the following equation yields good results for image intensity when using the gradient shading algorithm:

$$I = \frac{(I_{max} - I_a)\,(D - d)\,(\cos\theta)^p}{D} + I_a \tag{3.18}$$

where I_a is the ambient light, d is the distance to the surface from the light source, D is the distance at which illumination falls to zero, p is empirically determined (0.2 is suggested), and θ is the angle between the incident light and the normal to the surface. Gordon and Reynolds note that a z-buffer class algorithm need not be used for hidden-surface removal since any hidden-surface removal algorithm can be used so long as the distances from the observer to each pde on the visible portion of the object surface are retained.

The critical step in the above procedure is determining the value for ∇z at the point z(x,y). The value for ∇z can be found using:

$$\nabla z = \left(\frac{\partial_z}{\partial_x}, \frac{\partial_z}{\partial_y}, 1\right) \tag{3.19}$$

However, to determine ∇z one must first determine two derivatives numerically. The solution proposed by Gordon and Reynolds is to take a weighted average of the forward and backward differences as an approximation to the desired derivatives. The procedure for calculating $\frac{\partial_z}{\partial_x}$ is presented below, the same steps are followed for calculating $\frac{\partial_z}{\partial_y}$. Given an N x N z-buffer and the distances $z_{i,j} = z(x=i, y=j)$ for $(1 \le i \le N, 1 \le j \le N)$, $\frac{\partial_z}{\partial_x}$ can be approximated by using a weighted average of the forward and backward differences. The backward difference (∂_b) is found using.

$$\partial_b = z_{[i,j]} - z_{[i-1,j]} \qquad (2 \le i \le N\,,\ 1 \le j \le N) \tag{3.20}$$

The forward difference (∂_f) can be determined by:

$$\partial_f = z_{[i-1,j]} - z_{[i,j]} \qquad (1 \leq i \leq N-1 \,,\, 1 \leq j \leq N) \qquad \textbf{(3.21)}$$

A weighted average of these two differences is used so that the value used for the gradient can be adjusted according to the magnitude of the difference. If W_b and W_f represent positive weights applied to ∂_b and ∂_f respectively, then $\frac{\partial_z}{\partial_x}$ can be determined as follows:

$$\frac{\partial_z}{\partial_x} \cong \frac{(W_b * \delta_b) + (W_f * \delta_f)}{W_b + W_f} \qquad \textbf{(3.22)}$$

Small differences are weighted heavily and vice-versa. A key feature of this technique is that there are no normals or neighbor codes to be recalculated each time the orientation of the scene is changed, which is an advantage if a real-time display rate is desired. Gordon and Reynolds report that this technique produces good results even without employing smoothing and anti-aliasing, but that this method fails in portions of the scene where the slope is rapidly changing.

3.7.4 Normal-based Contextual Shading.

Normal-based contextual shading, described in [Che84a], calculates the shading value for a pde based on the surface normal of the visible face for the pde and the normals (context) for the surrounding pdes, which together form a neighbor code. The neighbor codes are used to approximate the surface normals for each face of a pde. The algorithm assumes that some form of surface-recognition procedure has been performed prior to beginning the shading operation, allowing us to determine the context for each face at no additional computational cost. Clearly, each visible face of the object must be examined once. The performance of the algorithm is better than might otherwise be expected expect since its computations are limited to those faces of each pde which are visible by the observer . This is accomplished by simply assigning each face of each pde to a direction table during the image-segmentation process. As a result, run-time increases linearly with increases in 3D image resolution.

Briefly, the process works as follows (see [Che84a] for details). The algorithm is based on the observation that acceptable shading of a pde face can be achieved using only the orientation of the face <u>and</u> of the four pde faces adjacent to it, which comprise its local context, to estimate the surface normal at the center of the face. Normal-based contextual shading takes two other factors into account. First, pde faces further from the observer, who is assumed to be the source of light, are shaded darker than faces closer to the observer. Second, faces nearly parallel to the direction of light reflect less light than those perpendicular to the direction of light, and so are shaded darker. For any edge of a face, the face adjacent to that edge can have one of three possible orientations. If each face to be shaded is assumed to be surrounded by four other faces, there is a total of 81 possible arrangements of adjacent faces. The actual orientation of the surrounding faces is described with a neighbor code, which is maintained along with the intensity data for the face. The algorithm is applied in turn to each visible pde face in the object being imaged. Once the surface normal is estimated, the shading at the point is calculated using the following procedure. If θ represents the angle between the estimated normal at the point P and the incident light, then N (θ) can be calculated using:

$$N(\theta) = \cos\left(\frac{\theta}{n}\right)^{p} \qquad \textbf{(3.23)}$$

where p=0.6 and n=2. Using this result, the shading assigned to a point P, S(P), is determined by:

$$S(P) = \left[\frac{M-L}{2R} * (R - d) * N(\theta) + L\right]\binom{M}{L} \qquad (3.24)$$

where $[v]\binom{M}{L} = L$ if $v<L$, v if $L<v<M$, and M if $M<v$.

If d represents the distance of P from the light source, and M, L, and R are user-defined (values of M=255, L=30, and R=radius of the smallest sphere which encloses the smallest rectangular box that encloses the boundary surface are recommended for a system with gray level values bounded by zero and 255).

3.7.5 Gouraud Shading.

Gouraud shading [Gou71] was developed to limit Mach band effects which can occur in an image when a single surface normal is applied across a polygonal face. The model assumes that an illumination model has been used to determine the intensity at each polygonal vertex. Vertex normals can be estimated using the techniques in gradient or normal-based contextual shading, but better results are obtained when the normals are computed directly from polygon plane equations or estimated from the averaged cross product of the edges which terminate at the vertex. A bilinear interpolation based on the intensity values computed for the vertices is then performed to determine the intensity for each pixel lying along the polygon edge connecting the vertices. Intensity values for pixels lying along scan lines connecting polygon edges are obtained by a bilinear interpolation of the intensity values found at the point where the scan line intersects each polygon edge. While the Gouraud model can be employed under a global illumination model, Rogers [Rog85] recommends using a diffuse shading model for the best results. The increase in run-time for this algorithm increases linearly with the increase in 2D resolution.

3.7.6 Phong Shading.

Phong shading is the final shading technique presented in this overview. The following discussion is based on [Pho75] and [Fol83]. Like normal-based contextual shading, it operates in object space, and so has the same running-time growth as normal-based contextual shading. However, of the five techniques, Phong shading gives the highest quality results, albeit along with the highest computational cost. Phong shading has not found wide use in medical imaging applications. In fact, no machine has been developed or proposed which uses Phong shading as its primary shading technique but it is offered as an option on a few medical imaging machines. This is due in large part to the amount of data involved in medical imaging applications and to the large computation cost incurred when Phong shading is used. While the increase in run-time is linear with the increase in 2D resolution, in the medical imaging environment the increase in image quality is not worth the computational cost that must be paid.

Phong's approach to shading is based on normal-vector interpolation. The method proceeds by interpolating the surface normal vector, $\bar{N}$, across the visible span of a pde on a scan line. The interpolation is based on the starting and ending normals for the span, which are themselves formed from interpolations along pde edges from the pde vertex normals. The pde vertex normals are computed directly from the image data. Once $\bar{N}$ is determined for each point on the scan-line, an illumination model, such as (3.12) is used to determine the intensity value assigned to each pixel along the scan-line.

3.8 Ray Tracing.

Ray tracing is a brute force technique for finding the visible surfaces within a scene. Ray tracing operates by following the path taken by rays of light originating at the observer through screen space and on into the scene and then assigning a value to the intersected screen space pixel based on the material(s) encountered by the ray. The appeal of ray tracing comes from its ability to model complicated reflection/refraction patterns and transparency, thereby providing "lifelike" images. Although ray tracing techniques are not employed within the research reported later in this dissertation, a brief discussion of ray tracing operations is warranted for two reasons. First, ray tracing is the "gold standard" for image rendering in terms of image quality, although the time required to form the image is extremely long. The dissertation would be incomplete without some reference to the body of work related to ray tracing and overcoming its computational expense. Second, the future work section of Chapter VII discusses a ray tracing implementation as a follow-on endeavor to my research, and a brief discussion here provides a framework for that presentation.

The seminal work in ray tracing is the paper by Whitted, [Whi80], which reported a ray tracing scheme which incorporates a shading model into the ray tracing image rendering calculations, with thereby producing high-quality, photo-realistic images. Ray tracing operates by projecting rays into image space from screen space. Whenever a ray intersects a surface, a secondary ray is traced from the intersection point to each point light source illuminating the scene. The tracing of this secondary ray determines if the object surface is in shadow with respect to each of the light sources. An object is in the shadow of an object with respect to a given light source if the secondary ray intersects any objects between the surface and light source. Additional rays are also spawned at the original object surface intersection point to account for reflection and and refraction, with the parent - child relationship between these three rays stored within a tree. These rays are then propagated further into image space, where they can in turn intersect with other surfaces, generating additional refraction, reflection, and shadow rays. Each generated ray is traced until it either intersects an object or leaves the image space volume. When all rays have been traced, the tree is traversed in reverse depth order, with a shading model (Whitted uses Phong's[1]) applied at each intersection to determine the intensity of the ray at the tree node. The computed intensity is attenuated by the distance between the parent and current child node, and then used as the input to the parent's intensity calculations as either reflected or refracted light.

To render anti-aliased images, Whitted projects four rays per pixel, one from each corner, and averages them if they are nearly equal. If the values have a wide variance, the pixel is subdivided into four quadrants and additional rays cast from the corners of the new quadrants. Quadrant subdivision continues until the minimum variance criteria is met, at which time the intensity of the pixel is calculated from the area weighted sums of the quadrants and sub-quadrants formed within the area of the pixel. Whitted recognized the computational expense of the intersection testing the algorithm requires and so used a regular-shaped bounding volume (a sphere) to enclose each object in order to accelerate intersection test processing. Because it is computationally simpler to determine if a ray intersects a regularly-shaped volume instead of the possibly convoluted surface of an object, the bounding volume test pays off in reduced image rendering time. Object - ray intersection tests are performed only if the ray intersects the object's bounding volume, otherwise the ray continues on its journey through image space.

Since Whitted's paper, research on improved methods for ray tracing has concentrated on reducing the computational cost of ray tracing with additional efforts

[1]Any other illumination model may be employed, such as the Cook-Torrance model or the Cook-Whitted model, as long as the model is based upon geometric optic concepts and not radiosity.

directed toward improving the photo-realism of the rendered image. Because of its relevance to my work and its current popularity due to its efficiency, only the tree-based acceleration techniques are described here. Over the next few paragraphs, the significant work in ray tracing acceleration and improved image quality is briefly summarized.

Dippe', in [Dip84] presents a technique for subdividing object space into regions with approximately equal workload within a 3D array of independent processors. The algorithm begins with an approximately equal subdivision of object space among the processors, and successively refines the subdivision as each processor determines its workload as processing runs accumulate. The basic idea within the algorithm is for heavily loaded processors to offload their workload onto lightly loaded adjacent processors by performing region subdivision among neighboring processors. The advantage of this technique is that it achieves a balanced workload among processors, but the image must remain static in order for the algorithm to work.

Glassner, see [Gla84], presents a technique for employing an octree to reduce the image rendering time required for ray tracing. Glassner's approach begins by recursively dividing object space into successively smaller, unequal volume, voxels with each voxel containing a complete description of each object which falls within it. The octree is used to maintain a record of the spatial relationships between the voxels, with successively smaller voxels assigned to deeper nodes within the tree. A terminal voxel is formed whenever the number of objects within a given volume falls below a specified number. To speed up image rendering, each node in the octree is assigned a number, based on a concatenation of the complete number of the parent node with the number of the child (1-8). This procedure yields a value of one for the root node, 11 - 18 for the root's eight children, and 111 - 118 for the children of node 11. Once the octree is formed, image rendering commences. As each ray is moved through image space, its path through each voxel is tested against the location of each object assigned to the voxel. If the ray intersects an object, the color of the first object hit is returned as the value of the ray, otherwise the ray proceeds on to the next voxel (ie. leaf of the octree). Movement between voxels is accomplished by moving the ray to a point guaranteed to lie within the next voxel along the ray and using the coordinates obtained by this movement to generate the number of the target voxel. The number of the target voxel is then hashed to determine its location in memory and accessed for its object list.

Glassner's recent work, [Gla88], combines the use of bounding volumes with spatial subdivision. His approach is to build an octree, with the depth termination criteria based upon number of objects per leaf, leaf volume, or density ratio. Each leaf contains a complete description of all the objects within its volume. Once the octree is built, then a tight bounding volume is defined for each node of the octree such that all the objects for the node are encompassed within the bounding volume and the bounding volume does not extend beyond the limits of the node's corresponding voxel. Objects which extend beyond a node's limits are divided and assigned to the appropriate nodes. These two steps produce a tree of bounding volumes, and combine the intersection test efficiency of bounding volumes with the non-overlapping hierarchy obtained with a spatial enumeration approach. As rays are cast into the space, each ray is tested in turn for intersection with the bounding volumes along its path until intersection is achieved. Once the ray intersects an object, the ray value is calculated and returned. The octree structure is used to guide propagation of the rays from bounding volume to bounding volume.

Akira Fujimoto et. al., in [Fuj86], describe an an octree-based approach to accelerating the performance of ray tracing algorithms. The approach rests upon two developments, SEADS (Spatially Enumerated Auxiliary Data Structure), which is used to store the tessellation of object space into voxels. Each leaf node in SEADS contains a description of the object which falls within the leaf. Basically, then, the SEADS structure encloses each object with a parallelpiped-shaped bounding volume and captures the spatial relationships between these volumes within the octree structure. Unlike other approaches, the SEADS structure operates independently of the shape of the objects in object space, ie.

no attempt is made to fit nodes into a "smooth" match of the object surface. The other innovation is the use of the 3DDDA (3D Digital Differential Analyzer) as a tool for octree traversal. The 3DDDA is used to trace each ray through space and allows direct movement in object space between leaf nodes of the same parent without movement up the tree to the common parent. Movement between leaf octree nodes is facilitated by an octant numbering scheme which permits direct movement between leaves of a single parent branch and indicates the parent branch to be explored next in cases where vertical movement up the tree is required. The authors claim that their traversal method is 13 times faster than the method described in [Gla84].

Marc Levoy, in [Lev88], [Lev89a], and [Lev89b], uses an octree coupled with adaptive termination of ray propagation to accelerate geometric volume rendering by ray tracing through a gel-like representation of a volume. In Levoy's approach, a ray's value is determined from the sum of the rgb values for all the voxels it intersects divided by the sum of the α-channel values within those voxels. The octree is used to guide ray propagation through space, as in the other approaches described above. Adaptive ray termination is accomplished by calculating the ray's value as it proceeds through space and stopping the ray when the change in ray value between voxels is less than a predetermined ε. This concept has been extended, by building on the work reported in [Ber86], to allow for successive refinement of the image by subsequent casting of additional rays using a finer casting grid. To limit the refined image rendering time, rays are only cast for pixels of high complexity, which is measured by the difference in color value across the face of the pixel, when the user is not interacting with the display. For pixels with a small difference across the face, no additional rays are cast. Pixels with a large difference are subdivided (as in [Whi80] and recolored by casting additional rays. The process of continued subdivision continues for successive renderings until the difference between adjacent areas falls below the difference criteria. Additional refinement can be forced by lowering the difference criteria for successive renderings.

Kaplan, in [Kap87], describes another tree-based approach to reducing the rendering time required for ray tracing images. The approach he outlines first forms a BSP (binary space partitioning) tree, as described in [Fuc80], of object space down to the level of unevenly sized but equal content small cubes, with each terminal cube containing information describing the objects which intersect it. The use of a BSP tree permits adaptive subdivision of space into cubes falling in front of and behind a set of slicing planes, with the plane location set so that approximately equal complexity[1] is obtained on each side of the plane. Once the BSP tree is formed, ray tracing begins. Each ray is first intersected with image space, thereby giving the ray an image-space origin within a terminal cube. The BSP tree is then traversed until the box which the ray intersected is located using the values for the slicing planes which were inserted into image space at each node of the tree during tree formation and retained for this use. Once the correct box is located, the objects within the box are tested for intersection with the ray. If there is an intersection, the value of the ray for the closest object intersection is returned, otherwise the ray is extended to the next cube, and the BSP tree re-traversed. To improve the performance of his algorithm, Kaplan recommends the use of bounding boxes around the objects in each terminal node in order to reduce the cost of intersection testing.

Catmull's presentation, in [Cat84], sketches an algorithm for independent visible surface determination at each pixel. The algorithm assumes that texturing and shading are accomplished in a preprocessing step for all objects in the scene and that all objects in the scene have been reduced to polygons. Once preprocessing is accomplished, the objects are sorted on x' and y' to determine the screen space coverage of each polygon, and the polygons are assigned to each pixel for further processing based on pixel coverage. At each pixel, a sorted list of polygons is created using a head sort procedure. This sort operates by first finding the polygon with the smallest z' value and placing it on the object

[1]Complexity is measured by the number of objects within the volume.

list for the pixel. The next step in the sort locates all other polygons which overlap this object or any other object on the list in the z'-dimension. When the sort concludes, a gap has been created between the objects at the head and tail of the list, with the objects at the head having the characteristic that they all overlap in the z' dimension. Only the head of the list is examined in all further processing. If the polygon at the head of the list completely covers the pixel, the pixel is assigned the polygon's color. Otherwise the sorted list is dispatched to a filter routine where the pixel area of each polygon in the head is calculated and the pixel color is determined by a area-coverage weighted sum of the object colors. An advantage of this method is that the computations required to color each pixel can be easily partitioned among the processors in a multiprocessor because each pixel has all the information required for coloring, and communication with other processors to acquire additional shading information is not required.

Carpenter, Cook, and Porter, in [Coo84a], [Coo84b], and [Car84], describe enhancements to the basic ray tracing algorithm in [Whi80]. [Coo84a] describes a tree-structured shading model for flexible shading based on each object's surface characteristics and the surfaces of the objects reflected in it. [Car84] describes a scanline-based algorithm for hidden-surface removal, the A-buffer, to correct for the aliasing problems associated with the z-buffer hidden-surface removal algorithm in a ray tracing environment. The algorithm requires that the objects in the scene be converted into polygons and sorted into scanline order, with the polygons composing a single object identified by an object unique tag. To provide an anti-aliasing capability, each pixel is logically composed of a 4 x 8 grid which is used to represent subpixel polygons. The A-buffer operates on two different data types, pixel-structs, which store either the pixel color or a linked list of polygons which cover the pixel, and fragments, which are polygons clipped to pixel boundaries. Each fragment defines the pixel coverage it has in terms of the 4 x 8 subpixel grid. Pixel color is determined by determining the color of each fragment which covers the pixel. If the foremost fragment completely covers the pixel, the pixel is assigned the fragment color.

Otherwise, the fragment's 4 x 8 grid is used to determine which portions of the 4 x 8 subpixel grid are covered. The obscured area is assigned the fragment color and the remaining pixel area is colored by continued examination of the polygon list until all 32 positions are colored in the grid. The final pixel color is obtained by computing the sum of the fragment colors weighted by the area coverage of each fragment.

Distributed ray tracing, described in [Coo84b], is a technique for incorporating motion blur, penumbras, fuzzy reflections, and translucency. The algorithm requires the distribution of rays over time and other image-rendering parameters, such as apparent light motion and camera lens area. The key notion to the approach is that once the proper number of rays to achieve anti-aliasing is determined, then these rays can be distributed over the several image parameter dimensions instead of anti-aliasing the image with the same number of anti-aliasing rays in each dimension. Basically, no additional rays are needed beyond those required for oversampling in normal space, so realistic motion and lighting effects are achieved at little additional computational expense.

Upson and Keeler's V-buffer, described in [Ups88], is a technique for volume rendering which employs two separate methods for generating an image, a voxel-by-voxel method and a ray tracing method. Their approach selects one of the algorithms based on the image quality and rendering speed desired, and then the selected algorithm processes the entire volume to render the image. The ray tracing method casts rays from each pixel into image space and propagates the ray until the ray leaves the image volume or the ray opacity (α channel) reaches a value of one (completely opaque). Within each voxel, the color evaluation function is evaluated at the voxel face and then stepped through the voxel with additional evaluations performed at each stepping point. In order to terminate ray propagation adaptively, color values are integrated as the ray is stepped along.

The other method described by the authors is object-space oriented and processes the volume one voxel[1] at a time. The algorithm they describe is a *voxel-by-voxel, front-to-back* traversal of the volume, with all voxels in a single plane processed before any voxels in the next deeper plane are examined. Within a plane, the voxels are processed in order from the closest voxel in the plane concentrically out to the furthest voxel(s). For each voxel, the trilinear interpolation coefficients for shading, texture, and scalar functions are computed and a bounding box for the voxel determined. The bounding box, in turn, is used to indicate the pixels the voxel projects onto by computing bounding box - scanline intersections. For each scanline intersected by the bounding box, the voxel is intersected with the scan plane and the resulting convex polygon broken down into five spans or less. The pixels on the scanline are then colored using the color and texture information for the appropriate span, with the spans themselves processed in front-to-back order. The authors point out that this algorithm leans to incremental refinement of the image because much of the image is computed early in the rendering process, with additional information added to the picture as the algorithm progresses deeper into the volume. In their analysis of the two algorithms, the authors point out the tradeoffs in the use of either algorithm. The ray casting method requires more memory than the voxel approach, 16Mbytes for a 64^3 voxel domain vs 1.3Mbytes for the voxel approach. Each method can incorporate anti-aliasing, but the authors state that they attain acceptable image quality without its use. The ray casting method is most efficient in a volume with many opaque voxels, while the voxel approach is most efficient within transparent volumes. Their ray casting method is best suited to a uniprocessor architecture because it integrates pixel values as it progresses, whereas the voxel algorithm is better suited to a multiprocessor implementation since the voxels can be processed independently.

[1]The authors use the term cell instead of voxel. For consistency with the remainder of this chapter, their approach is described using voxels with the understanding that the voxels may be quite large.

3.9 Hidden-Surface Removal.

Hidden-surface removal algorithms attack one of the more difficult problems in computer graphics, determination of the surfaces which are visible/invisible from a given location in image space. Hidden-surface removal algorithms fall into three main types, scan-line based, depth-sort, and z-buffer. All three types display linear growth in computation time with increase in 3D scene resolution. Below, six hidden-surface removal algorithms are examined, a scan-line based algorithm, a depth-sort algorithm, a z-buffer algorithm, and three variations on the depth-sort algorithm, the back-to-front readout algorithm, the recursive back-to-front readout algorithm, and a front-to-back readout algorithm.

3.9.1 Scan-line Algorithms.

Scan-line algorithms operate on edges in the scene, not on the cuberilles themselves. Visible portions of the scene are identified scan-line by scan-line of the raster-scan display. The geometry for the scanline algorithm (with X', Y', and Z' being image-space coordinates) is presented in Figure 3.6.

The scan-line algorithm can be used in both the cuberille and octree data models, but since it is relatively simple to determine which polygon out of several is visible at any time, this algorithm is more suitable to the octree model than to the cuberille model. One drawback that this algorithm faces is that in order for it to be used, one must first determine the plane equations for each of the cuberilles which make up each object in the scene, which can be computationally costly. Although the run-time of scan-line algorithms increases linearly with 3D resolution increase, in the medical imaging environment under

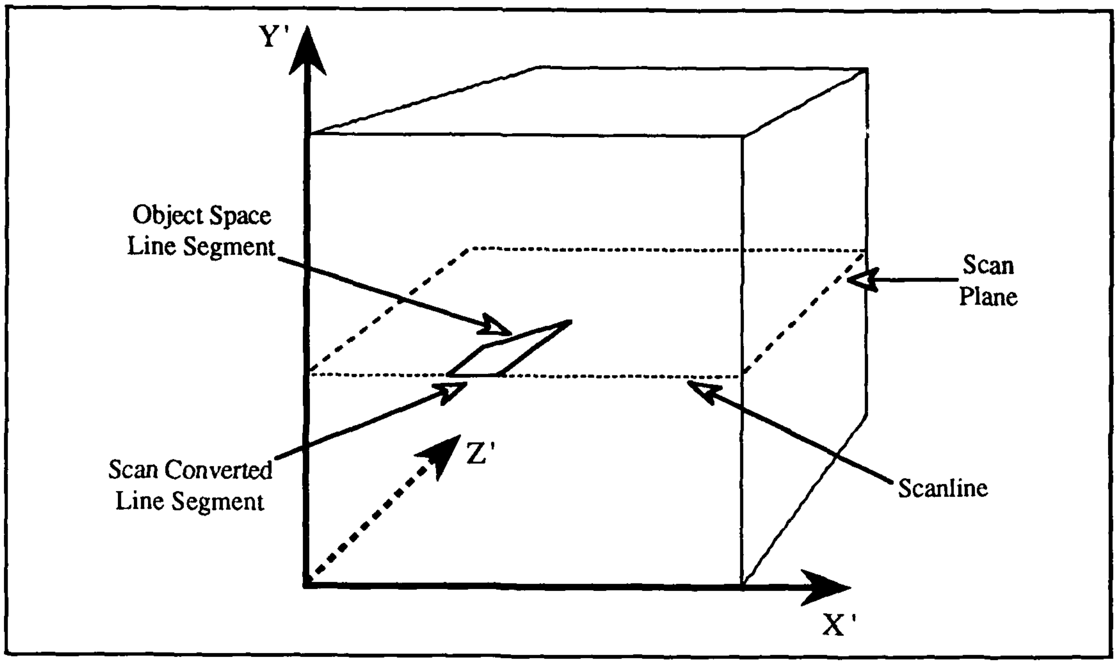


Figure 3.6: Scan-line Algorithm Geometry.

the cuberille data model the algorithm is not appropriate for hidden-surface removal because of its computation costs.

Before turning to an example of a scan-line algorithm, a short discussion of the theoretical underpinnings of this class of algorithms is required. All of the scan-line algorithms reported in the literature exploit the properties of scan-line coherence and of edge coherence in an image. Scan-line coherence is the property of a displayed object wherein the appearance of the object does not change by much when moving along a scan-line from pixel j to pixel j+1. Edge coherence is a property of a displayed object wherein many of the edges of the object which intersect scan-line i will also intersect scan-line i+1. These coherence properties are exploited by examining the image for those cuberilles at which changes occur. The object is then rendered by connecting these cuberilles with straight lines.

This class of algorithms operates in the following general manner. In a preprocessing step, two tables are constructed, an edge table and a polygon table. The edge table is created anew for the entire scene for each new scene orientation. Each entry in the edge table corresponds to a non-horizontal edge of a polygon in the scene and contains the y-coordinate for each end of the edge, the x-coordinate associated with the smaller y-coordinate, the inverse slope of the edge, a polygon identification, and the x-increment, Δx. The entries in the edge table are sorted into buckets according to the smaller y-coordinate value for each entry. The entries within each bucket are sorted again based on the x-value for each entry and the inverse slope of the corresponding edge. The polygon table contains the coefficients of the plane equation for each polygon, shading/color information for the polygon, and an In/Out boolean flag used during scan-line processing. Once these tables are established, image production can commence. Clearly, there is a considerable amount of preprocessing to be done before image formation can begin, and this preprocessing must be re-accomplished for each new orientation of the scene.

The active edge table is created and maintained while the image is being formed. Its entries are maintained in increasing x-order. The first entries to be placed in the active edge table are those entries in the previously created edge table which have the lowest y value. The pixels in scan-line y are filled in according to the pairs of coordinates in the active edge table. Those entries in the active edge table for which $y=y_{max}$ are removed. For each remaining entry in the active edge table, the current x-value is replaced by $x + 1/\Delta x$. The value of y is increased, and any new polygons found in edge table bucket y are entered into the active edge table. The next scan-line is filled in, and the process repeats until all the entries in the edge table have been processed.

Since several polygons may cross any given scan-line, the polygon the algorithm is operating within must be known at all times. The In/Out boolean flag in the polygon table is used for this purpose. Since the edges in the table are sorted by increasing x-value, the polygon In/Out flag can be inverted each time an endpoint of an edge of the polygon is processed, thereby indicating activity by the algorithm within that particular polygon. If the algorithm is ever in more than one polygon at a time, then the plane equations for each polygon are used to determine which of the polygons is closer to the viewer.

3.9.2 The Depth-sort Algorithm.

The depth-sort algorithm simply requires sorting the polygons in the scene according to their distance from the observer's viewpoint and then writing them to the CRT display in order of decreasing distance (see [Fri85a] for an example). We again have a situation wherein the running time for the algorithm grows linearly with increases in 3D resolution, and any effort to significantly increase the resolution of the object is overwhelmed by rapidly growing computation costs. As an example algorithm in this class, the depth-sort algorithm developed by Newell, Newell, and Sancha is presented.

Their algorithm proceeds as follows. First, all the polygons in the picture are sorted according to the largest z-coordinate of each polygon. As the polygons are sorted, they are placed in the sorted polygon list. If the z-extents of any polygons overlap, these conflicts are resolved according to the following five tests. When any test succeeds, it is applied immediately. The five tests are: 1) Determine if the polygons' x-extents overlap, if the x-extents do not overlap, then the polygons do not overlap. 2) Determine if the polygons' y-extents overlap, if the y-extents do not overlap, then the polygons do not overlap. 3) One polygon is entirely on the side of the other which is further from the viewpoint. 4) One polygon is entirely on the side of the other which is closer to the viewpoint. 5) Determine if the projections of the polygons onto the screen overlap. If the final test fails, then the polygons do not overlap. The z-extent overlap conflict is then resolved by simply dividing one polygon by the plane of the other, thereby creating two new polygons. These new polygons are then placed in their appropriate place in the sorted polygon list.

Once all the polygons are sorted by their distance from the viewer they can then be read into the refresh buffer in order of decreasing distance. Since the polygons nearest the observer are converted last, they overwrite any polygons which they obscure. The resulting picture displays only those portions of the polygons in the scene which are visible from the observer's position. Algorithms which use the strategy of overwriting obscured portions of the image by visible portions of the image are classified as painter's algorithms. This strategy characterizes the approach taken by the last three algorithms discussed.

3.9.3 The Z-buffer Algorithm.

Z-buffer algorithms make use of the overwriting strategy described above. In addition, the z-buffer algorithm requires the use of two buffers. The refresh buffer, which is always required when forming a digital image, is used for storing CRT pixel intensity values. The other buffer is a z-buffer which is used to store the z-value of the cuberille that is currently mapped to each CRT pixel. The z-buffer is initialized to the largest representable z-value, and the refresh buffer is initialized to the background value. Each cuberille in the scene is then scan converted. The conversion yields a depth z(x,y) at screen position (x,y). If the newly computed value for z(x,y) is less than the current value of z(x,y) in the z-buffer, then the new z(x,y) replaces the old z(x,y) in the z-buffer and the intensity value for the cuberille at z(x,y) replaces the intensity value at screen position (x,y). Alternatively, whenever a value of z(x,y) is found that is less than that stored in that (x,y) location in the z-buffer, then the cuberille that was just scan converted must be closer to the observer than the cuberille stored at that (x,y), and so the intensity value for the new cuberille should be displayed. The z-buffer algorithm is one of the simplest algorithms to implement, but it suffers from one great liability that the other algorithms avoid, viz, its memory requirements are far greater than theirs due to the need for two buffers equal in size to the final screen (2D) resolution. So, in addition to having the same running-time growth as the other algorithms in this section , the z-buffer algorithm has dramatically increased memory requirements when compared to these same algorithms.

3.9.4 The Back-to-Front Readout Algorithm.

The next technique for hidden-surface removal is back-to-front (BTF) readout of the scene as discussed in [Fri85a]. The algorithm accomplishes BTF readout of the scene in a recursive manner which closely parallels hidden-surface removal operations on an octree (see Appendix B). However, BTF algorithms are more efficient than comparable hidden-surface removal operations in an octree in that they do not require the construction of an octree, which serves to reduce the amount of preprocessing required to form an image. The algorithm simply accesses the entire data set in back-to-front order relative to the observer. In most circumstances, this forces an initial sort of the data set by z-value before the algorithm can be employed. However, in the medical imaging environment this requirement levies no additional computation cost since the voxel array produced by the CT scanner is already sorted. Therefore, the only work to be done is to read out the cuberilles in correct BTF order. This algorithm is even simpler to implement than the z-buffer algorithm and requires less space since there is no z-buffer to maintain. Note that the cuberilles can be read out in order of increasing x or y or z. The choice of the fastest changing index is arbitrary. (We know that the choice is arbitrary because if part of a voxel with coordinates (x,y,z) is obscured by part of a voxel with coordinates (x',y',z') then it must be true that $x \leq x'$ and $y \leq y'$ and $z \leq z'$, so that projecting (x,y,z) before (x',y',z') still produces a correct image.) Before reading out the cuberilles, the scene is reoriented according to the observer's viewpoint by rotating the scene as described above in the section on geometric transformations. Once the scene is correctly oriented, the cuberilles in the scene are read out in back-to-front order and mapped onto the image display. As is the case for the other algorithms in this section, run time increases linearly with 3D display resolution increase.

A comparison of the z-buffer and BTF algorithms reveals that the BTF algorithm is clearly superior. The only advantage the z-buffer algorithm possesses is that it does not require that the cuberilles be pre-sorted by z-value, since the sorting occurs implicitly at run-time. However, because of the additional computations required to determine z-values, the z-buffer algorithm does take longer to produce the final image than the BTF algorithm. In addition, the z-buffer algorithm requires twice as much space as the BTF algorithm. Clearly, if rapid image formation and minimal space are important performance objectives, and the scene is pre-sorted by z-value, then the back-to-front algorithm is the algorithm of choice.

3.9.5 Reynolds' Recursive Back-to-Front Algorithm.

The recursive back-to-front algorithm presented here was first described in [Rey85]. Like the back-to-front algorithm, the recursive back-to-front algorithm exploits the fact that medical image data are spatially pre-sorted. The recursive BTF algorithm is related to the painter's algorithm, but differs in that the cuberilles are not necessarily read out in decreasing distance from the observer. We first discuss the theoretical basis for the algorithm, then briefly describe its operation. Further details can be found in [Rey85].

Note that for any orientation of a scene, there exists a readout sequence such that cuberilles read out first can not obscure those read out last. This is because the cuberilles read out first are at the back of the scene (have a low priority) and those read out last are at the front of the scene (have a high priority). A notable feature of this property is that it can be applied recursively to the scene. This feature can be exploited to accomplish hidden-surface removal as follows.

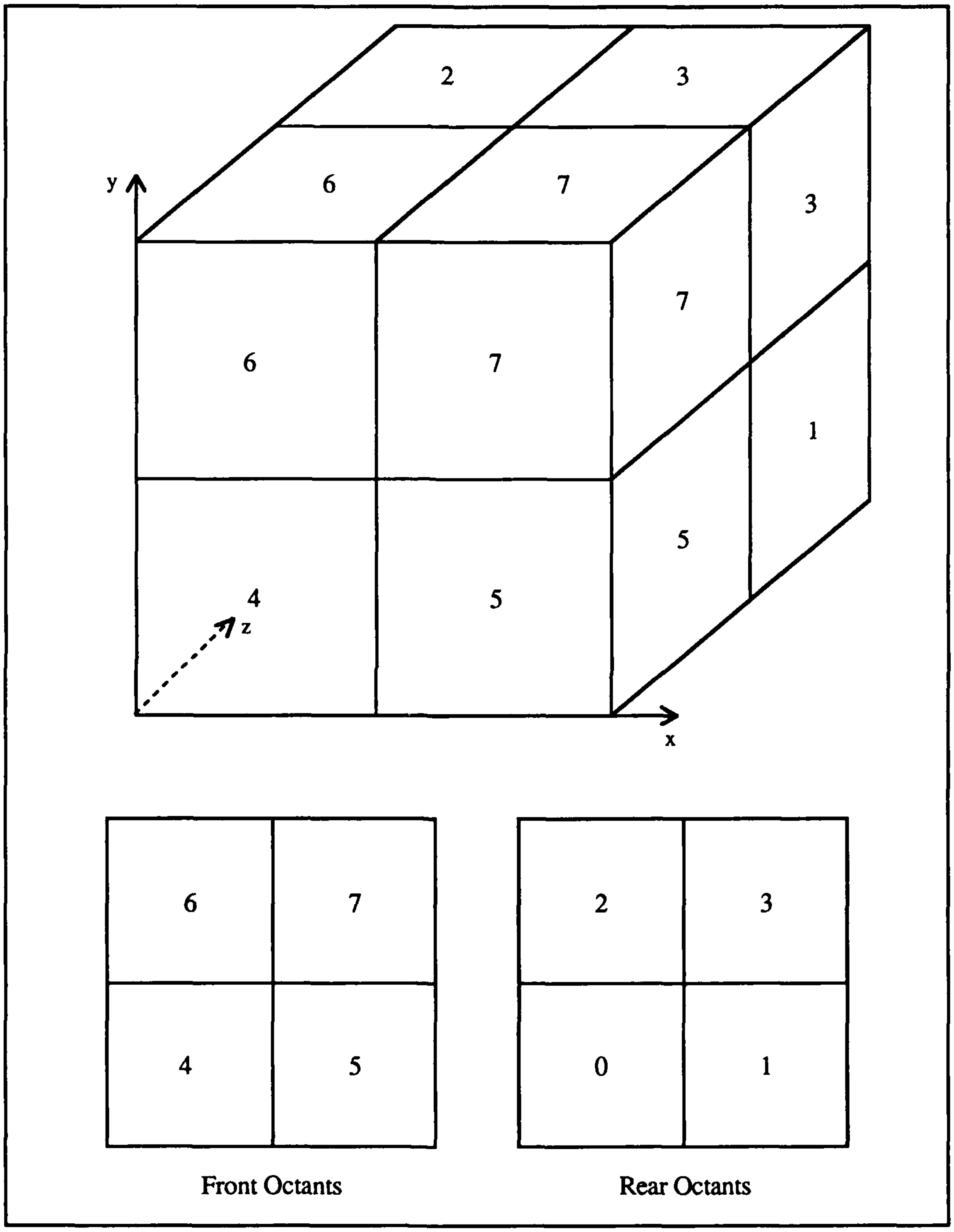


Figure 3.7: Octant Numbering Scheme.

Divide a scene into octants and number the scene octants as depicted in Figure 3.7. Now, divide each of the scene octants into eight suboctants and number the suboctants using the same scheme used for the scene octants. Continue this octant subdivision process down to the individual cuberille level, when it must stop. For any given observer position, a BTF readout sequence can be determined and expressed using the scene octant-numbering scheme. For example, given the orientation of the scene in Figure 3.7, the BTF priority readout sequence is octants 0,1,2,3,4,5,6,7 in that order. Because of the recursive manner in which the scene was decomposed, the BTF priority readout sequence which worked for the scene octants can be used as the BTF priority readout sequence for the suboctants, and the suboctants of the suboctants, etc. Of course, in order to accomplish hidden-surface removal all of the suboctants of an octant must be processed before the siblings of the octant are processed. Therefore, once the BTF readout sequence for the scene octants is established, all of the information required to correctly read out the cuberilles in the scene in BTF order is at hand. In order to form the image, all that remains to be determined is the coordinate mapping from object space to image space. The mapping is implicitly contained in the BTF readout sequence and is retrieved as described below.

The key to performing the mapping from object space to image space lies in the realization that the decomposition described above can be applied in both object space and image space and then "unwound" in p ¨allel. The mapping process is simplified if the scene octants are represented by the coordinate values of their center cuberille, and the display quadrants by the coordinate values of their center pixel. In each scene octant, represent the center coordinate for each suboctant as the sum of its offset from the center of the scene octant and the center coordinate for the scene octant. In each suboctant, represent each sub-suboctant's center coordinate as a sum of its offset from the center of the suboctant, the suboctant's offset, and the scene-octant's coordinate. So, at each step of the decomposition, there is a sum of N terms, where N-1 is the current depth of the recursion. Apply this coordinate decomposition to successively smaller octants until the cuberilles themselves are labeled. Perform the same decomposition in image space until each the coordinates for each pixel are expressed in terms of a sum of quadrant-center offsets. To clarify the decomposition process, an example of image space decomposition is provided below. Refer to Figure 3.8.

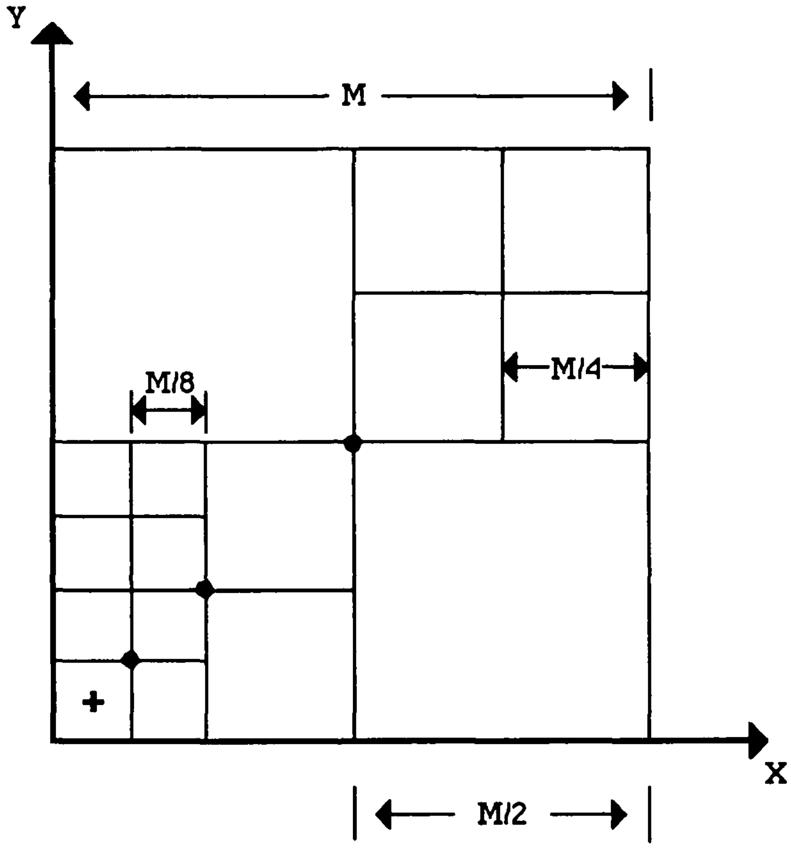


Figure 3.8: 2D Coordinate Transformation.

If the origin is placed at the lower left of the scene, the coordinate values for the pixel marked by the + are given by:

$$y_{+} = \frac{M}{4} - \frac{M}{8} - \frac{M}{16} \tag{3.25}$$

$$x_{+} = \frac{M}{4} - \frac{M}{8} - \frac{M}{16} \tag{3.26}$$

The coordinate values for the remainder of the pixels in the scene can be computed similarly using appropriate adds and subtracts when determining the coordinate values for successively smaller quadrants.

Hidden-surface removal for an arbitrarily oriented scene is accomplished as follows. First, permanently assign an octant number to each set of scene octant coordinates. Perform coordinate transformations on the scene by applying the geometric transformation operations described earlier in the chapter to the scene octant centers. The BTF readout sequence for the scene is obtained by sorting the resulting coordinate values for the scene octants according to z-value, which yields the correct BTF octant number sequence. Coordinate values for the successively smaller octants of the object and image space can be obtained using the coordinate decomposition procedure. Simply compute the coordinate offsets for the center of each suboctant in BTF order by taking the center coordinate values for each scene octant in BTF order and dividing them by 2^{N-1}. Then add/subtract the resulting offset to/from the coordinate center offset series previously computed for the center of the octant at the preceding level of recursion. If the object and image space resolutions are identical, the object space to image space mapping is accomplished by decomposing the two spaces in parallel, and assigning the cuberille CT number at each resulting object space coordinate to the pixel at the corresponding image space coordinate. The final image has all hidden-surfaces removed.

3.9.6 The Dynamic Screen Front-to-Back Algorithm.

The final hidden-surface removal algorithm to be examined is the dynamic screen algorithm presented in [Rey87]. This algorithm differs from those discussed earlier in three key regards. First, this algorithm begins its operation at the front of the data volume, necessitating a mechanism for keeping track of those pixels which have been written to.

Second, it does not operate directly on voxel data, rather a pre-processing step which transforms the data set into binary-valued contours which depict the surface of interest within the data must be performed. Third, the algorithm maintains a data structure, the dynamic screen, which is used to determining when volume processing can cease. As the concepts of front-to-back hidden-surface removal mirror those of back-to-front hidden-surface removal, the following discussion concentrates solely on the algorithm..

The dynamic screen technique employs binary-coded contours to describe a volume from different, viewer selected orientations. To provide the maximum amount of data compression, the contours are coded as binary values, with each contour represented by its start and end x-coordinates. The dynamic screen itself is a data structure used to store the location of unlit pixels, whenever a pixel is lit, it is removed from the data structure. The algorithm terminates when the last unlit pixel is removed from the dynamic screen.

Before image processing commences, the raw gray-scale voxel data is transformed into an auxiliary image formed from binary-contour data using thresholding or one of the other image segmentation algorithms described in Section 3.5. The algorithm processes the volume by reading in slices of binary-contour data one, at a time, in front-to-back sequence. The individual rows of the slice are then scan converted in front-to-back sequence. For each row in the slice, the object segments are processed in front-to-back order, with each segment merged with its corresponding scanline on the dynamic screen. As each object segment is merged, only the pixels which are not already lit, signified by their presence on the dynamic screen scanline, can be lit. When a pixel is lit, it is removed from the dynamic screen and the gray-scale value(s) of the object segment in the original image is written to the frame buffer. When the last pixel is lit, the algorithm terminates and the image in the frame buffer is displayed.

The segment merge process is the heart of the technique, and its operation is directly tied to the dynamic screen data structure. The dynamic screen is represented by an array of pointers, with the array equal in size to the y-dimension of the screen. Each pointer in the array is at the head of a linked-list composed of unlit segments on the scanline. Each linked-list element has three fields, the max and min unlit pixels within the the dynamic screen segment and a pointer to the next entry in the linked list. Merging is accomplished by starting with the first (in front-to-back order) object segment on the scanline and the first unlit scanline segment in the dynamic screen. The starting and ending points for the object segment and screen segment are compared. If the object segment falls outside of the screen segment, the next screen segment is examined. If the object segment falls within the screen segment, the max and min values for the segment are updated to reflect the addition of a new segment to the screen (with optional splitting of the screen segment if the object segment falls completely within the screen segment) and the next object segment is processed. To prevent aliasing artifacts when object segments fall across more than one scanline, Reynolds recommends scan-converting parallelpipeds instead of line segments and using a digital difference analysis computation to step along the path of the parallelpiped.

3.10 Chapter Summary.

This chapter has examined the important classes of algorithms used to form a simulated 3D image under the octree and cuberille data models. For the most part, the algorithms fall prey to rapid growth in computation costs as the resolution of the 3D scene increases. As the discussion in Chapters V and VI demonstrate, this growth can be curtailed for the front-to-back and back-to-front algorithms by taking advantage of volume coherence properties and a rapid pre-processing step. However, the need to accommodate this growth in computation costs, more than any other single factor, has influenced the design decisions made in the machines we examine in the next chapter.

CHAPTER IV
MEDICAL IMAGING MACHINES

4.1 Introduction.

This chapter builds upon the last chapter's review of graphics processing operations and data models in its survey of previous work in the medical imaging field. To highlight the challenges encountered in the design of a medical imaging machine, the chapter begins with a discussion of the facets of medical imaging which distinguish it from other graphics processing tasks. This is followed by a survey of the design and performance of eleven medical imaging machines. The following section discusses five research-oriented medical imaging machines followed by a section which discusses six commercial medical imaging machines. The focus of the description of each machine is on the innovations and qualities which set each machine apart from its fellows. The chapter concludes with a brief summary of the lessons learned by this survey.

4.2 Unique Aspects and Operational Requirements Of Medical Imaging.

There is no one aspect of medical imaging which distinguishes it from other graphics processing applications. Rather, it is the convergence of several factors which makes this field unique. Two well known factors that are closely related are computational cost and data volume. The typical medical image contains a large amount of data: the average CT procedure generates ten million voxels[1], MRI, PET, and ultrasound procedures produce similar amounts of data. A second factor is that the algorithms employed for medical imaging have great computational cost even at moderate 3D resolution.

Admittedly, other imaging applications, such as space exploration and oil field exploration, generate larger data sets and have considerable image rendering computational cost. However, these other large-scale applications lack one significant feature of the medical imaging environment: its direct and immediate impact on the daily lives of a large number of people. This impact derives from the medical imaging application itself, the results of the imaging process are used to diagnose disease and to make disease treatment decisions. The goal, then, of medical imaging is to provide accurate, timely diagnostic and treatment information to the physician rapidly enough so that the service is available to all the patients who need it. This goal, and the inflexible requirement for rendering accuracy, serves to distinguish the medical imaging field from other large-scale image rendering applications.

My survey of the medical imaging literature disclosed additional factors. First, there are no underlying geometric models which can be applied to medical image data to simplify it. A hard and fast requirement is that the organ, as it exists in the body, be accurately portrayed. Only a limited amount of abstraction from the raw data is permitted: diagnosis is based upon departures from the norm, and high level models reduce, or eliminate, these differences. Therefore, from both the clinical and medical imaging viewpoints, data models for organs, or organ subsystems, are irrelevant for image rendering purposes. A second factor is that the display is not static. Typically, users will want to interact with the display to perform digital dissection and rotation operations upon it. Third, a three-dimensional (3D) representation provides a potential increase in clinical

---

[1]Assuming forty 512 x 512 slices.

usefulness over a 2D representation because it can portray the scene from all points of view. The convergence of these factors sets medical imaging apart from all other imaging tasks.

Meeting the medical imaging quality requirements while performing the medical imaging operations rapidly is a difficult challenge. In fact, many of the requirements and operations work at cross purposes, and achieving one usually requires sacrificing performance in one or more other areas. To date, no single machine has met all objectives and implemented all the operations, and it is an open question whether all the objectives can be met in the same machine.

4.3 Research-Oriented Medical Imaging Machines.

The next two sections are devoted to describing eleven image processing architectures: The Ardent™ Titan™, AT&T Pixel Machine™ 900 Series, Farrell's Colored-Range Method, Fuchs/Poulton Pixel-Planes machines, Kaufman's Cube Architecture, the Medical Imaging Processing Group's (MIPG) machine, the Pixar™ Image Computer and Pixar II, Reynolds and Goldwasser's Voxel Processor machine, Robb's true 3D machine (the DSR), the Stellar® GS series, and the Sun™ TAAC-1™. Each machine is characterized by: 1) The overall machine architecture. 2) The processing strategy. 3) The data model used. 4) The shading algorithm used. 5) The anti-aliasing technique employed. 6) The hidden-surface removal algorithm employed. 7) The performance of the machine. 8) The image resolution supported. 9) The level at which the machine exploits computational parallelism. This section is devoted to describing the research-oriented machines, Farrell's Colored-Range Method, Fuchs/Poulton Pixel-Planes, Kaufman's Cube Architecture, the MIPG machine, and the Voxel Processor. Section four concludes the review with a description of the five remaining commercial machines. A summary of the key parameters for each of the eleven machines is presented in Table 4.1.

TABLE 4.1
Medical Imaging Machines

	Ardent Titan	AT&T Pixel Machine 900 Series	Farrell's Colored-Range Method	Fuchs/Poulton Pixel Planes 4	Fuchs/Poulton Pixel Planes 5	Kaufman's Cube Architecture	MIPG Machine	Pixar II	Voxel Processor Machine	Robb's True 3D Machine	Stellar GS2000	Sun TAAC-1
ARCHITECTURE	MIMD pipeline	Host, MIMD pipeline and SIMD array	Mainframe and workstation	Host , SIMD pipeline, and SIMD frame buffer	Host , MIMD processor pipeline, and multiple SIMD frame buffers	Host and Cube processor	CT scanner microprocessor	Host and SIMD array	Host and SIMD Pipeline	Mainframe and workstation	Host, MIMD pipeline and SIMD array	MIMD pipeline
OBJECT MODEL	Contour or voxel	Voxel	Voxel	Contour or voxel	Contour or voxel	Voxel	Voxel	Pixel	Voxel	Voxel	Contour or voxel	Point or voxel
SHADING	Flat, Gouraud, and Phong	Flat, Gouraud, and Phong	Back-to-front increase in lighting	Gouraud and Phong	Gouraud and Phong	Congradient	Distance, contextual, gradient, and Phong	Combination of light absorption, luminosity, and surface texture	Gradient shading	Not required	Distance, Gouraud, gradient, and Phong	Cubevu - Flat and Gouraud Bayvu -various weighted density functions along ray
ANTI-ALIASING	Ray tracing - stochastic sampling Volume - nearest neighbor	Piclib - supersampling at user defined size Baylib - adaptive stochastic	3 x 3 local average	4 x 4 local average	4 x 4 local average	None	Super sampling to double resolution	Supersampling or low-pass filtering	Super sampling to double resolution	Not required	3 x 3 filter	Trilinear interpolation
HIDDEN-SURFACE REMOVAL	Z-buffer	Z-buffer or backface surface removal	Overwrite unnecessary objects in scene	Z-buffer	Z-buffer	Front - to - back	Modified z-buffer	Back - to - front	Back - to - front	Z-buffer	Z-buffer	Z-buffer
RESOLUTION	1280 x 1024	1280 x 1024 or 1024 x 1024	1024 x 1024	512 x 512	1280 x 1024	512 x 512	256 x 256	1280 x 1024	512 x 512	128 x 128	1280 x 1024	1024 x 1024
PERFORMANCE	256 x 256 x 70 voxel volume @ 15 frames per second	256 x 256 x 60 voxel volume ray traced in 10 seconds	10 - 15 seconds per frame	30,000 triangles/sec, not real time	1-10 frames/sec.	2 frames per second (anticipated)	Slow	256 x 256 x 256 volume in minutes to hours per frame	25 frames p. second	27 frames per second	150,000 polygons per second. Medical image data not available	Cubevu - one slice in .039 seconds Bayvu - 256 x 256 x 32 voxel volume ray traced in 2 minutes
PARALLELISM	Moderate in pipeline	Pipeline: none Array: moderate	None	High in frame buffer	Low in pre-processor High in Renderer	Moderate - high in Cube	None	Low	Moderate	None	Moderate in pipeline	Low in logical pipeline

With three exceptions, the Fuchs/Poulton machine, Kaufman's Cube, and Reynolds and Goldwasser's Voxel Processor, these eleven machines do not provide real-time performance and are they are not special-purpose architectures. Instead, the machines are all general-purpose computers which use a combination of clever algorithms, rapid memory access techniques, and massive computational power to achieve acceptable elapsed image rendering time performance. The special-purpose machines proposed by Fuchs, Kaufman, and Reynolds provide near real-time image rendering rates by embedding the image processing algorithms in hardware, thereby sacrificing processing flexibility for image rendering speed.

Even in this brief survey of the eleven medical imaging machines it is evident that there is no one architecture or approach which commands the field. Rather, each machine has a suite of capabilities which distinguishes it from the other eleven. Note that these capabilities, such as software encoding of algorithms, use of general purpose architectures, apparent 3D, high speed, and high resolution, are not mutually exclusive. The goal of combining these capabilities into one medical imaging machine which provides a rich imaging environment for the clinician has yet to be achieved.

4.3.1 Farrell's Colored-Range Method.

Farrell's machine (see [Far84], [Far85], [Far86a], [Far86b], [Far87], and [Far89]) takes a processing approach that is different from the other machines discussed in this chapter. The machine has a high image processing speed, but the equipment cost is prohibitive for most users. The image processing system consists of a host and a workstation. The host is an IBM 370/4341. The workstation is an IBM 7350. A simplified diagram of the architecture is presented in Figure 4.1.

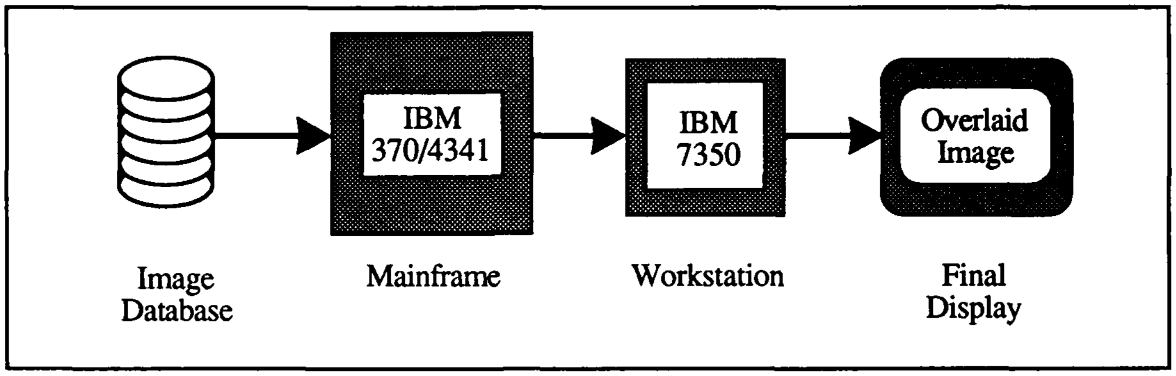


Figure 4.1: Farrell's Imaging System Architecture.

A significant problem faced by the designers of the machine was deciding how to divide the workload between the workstation and the host. The solution adopted was to assign the data file management, subimage selection, data smoothing and 3D rotation operations to the host. The remaining operations, oblique projection, data cutout, transparency, and color selection , were assigned to the workstation. Basically, the computationally intensive operations are placed in the host, and the remaining operations are placed in the workstation.

The processing strategy seeks to reduce the computational load by eliminating or simplifying the operations required to form a 3D image, as a result the colored-range method for back-to-front display was developed. This methodology does not require pre-proccessing of the data and is capable of rapidly displaying the data from an arbitrary point of view. In the colored-range method, 3D relationships of the constituent parts of the image are emphasized by image rotation and the use of color. To accomplish image rotation about an axis, the image is simply painted diagonally across the screen in the back-to-front order defined by the desired scene orientation. The required diagonal offset for successive 2D frames is achieved using x,y tables of screen coordinates for small rotational values, up to ± 40° from a head-on viewpoint. Rotation of the scene by larger amounts is accomplished by reformatting the data so that either the ±x, or ±y axis becomes the +z axis. Segmentation of the components of the image is accomplished by determining a discrete voxel value range which corresponds to each component of the image and then assigning a different color to each range. The relative depth of the parts of a given object is indicated by varying the lightness of the color according to the depth of the part, with more distant parts being darker. This technique has several advantages in addition to rapidly rendering the image. It allows different structures to be displayed simultaneously through the use of different colors and the relative size and position of structures can be interpreted as successive slices are overlaid. The data need not be continuous in 3D space, thereby eliminating the need for data interpolation. Additionally, the colored-range method displays accurate 3D images with just the use of simple logical and arithmetic functions in the processor. Finally, data management and storage functions and rendering operations which have a high computation cost are off-loaded from the display processor to the host. This action frees the display processor to perform the image rendering tasks, such as 2D rotation and pixel intensity calculation, directly related to forming the screen image.

4.3.1.1 Shading, Anti-aliasing, and Hidden-Surface Removal Techniques.

The object model used by Farrell is unique among the machines discussed in this survey, the raw voxel output is used without preprocessing. The voxels within the object(s) of interest are painted directly to the screen. Shading is accomplished by a back-to-front increase in color lightness. Anti-aliasing is accomplished using a 3 x 3 local moving average on the final image.

Farrell's machine performs hidden-surface removal without additional data structures such as z-buffers and octrees. Instead, the fact that the image data is spatially pre-sorted is used along with changes in intensity to accomplish hidden-surface removal using the painter's algorithm[1]. As each successive 2D image is overlaid, lighter colors are used to provide a 3D effect through depth-shading. This technique can be refined further by allowing the user to specify those objects (ie. density ranges) which are to be considered

[1]The painter's algorithm simply overwrites the distant portions of the scene by those objects which are closer to the viewer.

transparent. Thus, only the pixels within surfaces which are obscured by a user-designated non-transparent surface are overwritten. Transparent objects, those which are not of interest to the user, are ignored and their pixels are not allowed to overwrite pixels which are further away from the viewer. Processing the 3D digital image data in this manner creates gaps between structures in the scene which serves to further enhances the 3D effect.

4.3.1.2 Degree of Parallelism and Machine Performance.

There is little parallel operation in Farrell's machine. Parallel computations occur when the workstation is processing an image while the host is preparing a subsequent image for the workstation. On the other hand, images are generated at very high speeds. Farrell's image overlay technique permits 3D images to be processed and presented in a matter of 10 - 15 seconds. This speed comes from the elimination of preprocessing operations on the voxel based image and simplification of the shading and hidden-surface removal operations.

4.3.2 Fuchs/Poulton Pixel-Planes Machine.

At the opposite end of the parallelism scale from Farrell's Colored-Range Method is the Fuchs/Poulton Pixel-Planes machine, described in [Fuc77], [Fuc79], [Fuc80], [Fuc83], [Fuc85], [Fuc86], [Fuc88a], [Fuc88b], [Fuc89a], [Fuc89b], [Gol86], [Lev89b], [Piz84], [Piz85], and [Pou87]. This machine achieves high performance by assigning a one-bit processor to each pixel in the frame buffer. Two architectures for the machine have been described, Pixel-Planes 4 (Pxpl4) and Pixel-Planes 5 (Pxpl5). Pxpl4 is operational, with Pxpl5 to be operational in the fall of 1989. The discussion below concentrates on Pxpl4, as its concepts are carried through and expanded upon in the new machine, but a description of Pxpl5 is also provided. The goal of the Pixel-Planes architecture is to increase the rate at which pixels are displayed by performing graphics operations in parallel at the pixel level within a general-purpose graphics system. A diagram of the machine, from [Fuc88b], appears in Figure 4.2.

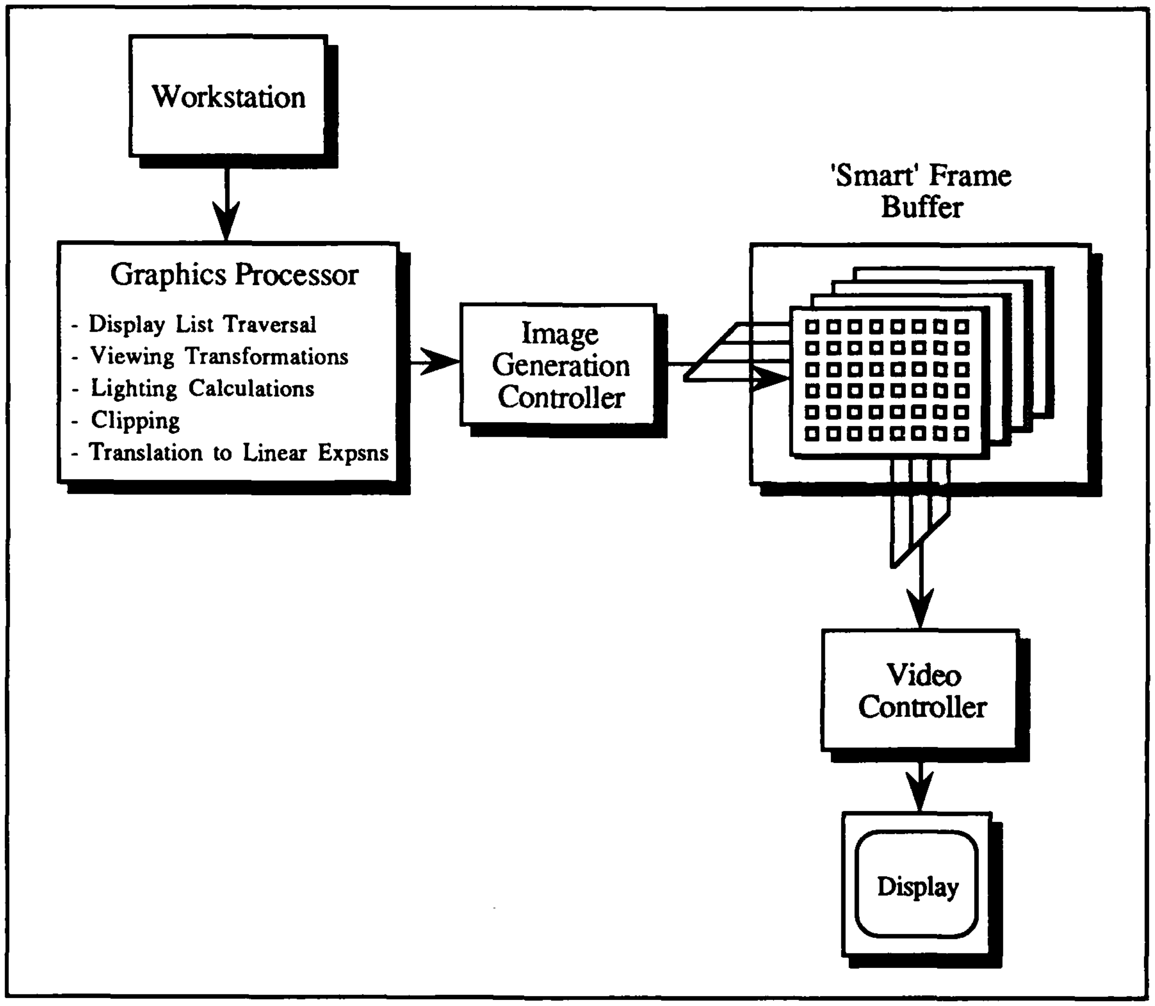


Figure 4.2: Pixel-Planes Machine Architecture (based upon [Fuc88b]).

The major components of the Pixel-Planes 4 machine are a workstation, graphics processor, image generation controller, and a smart frame buffer. The front end of the system specifies screen objects in a pixel independent format which the frame buffer memory chips then employ to generate the image. The high performance of the Pixel-Planes 4 machine is due to its "smart" frame buffer, which consists of 512 x 512 pixels by 72 bits per pixel. Within the smart frame buffer are the logic enhanced memory chips that are instructed to perform pixel-oriented tasks simultaneously at the pixel level. The host workstation can be any Unix-based system and must support the graphics libraries, microcode assembler, and compiler required by the Pxpl4 unit. The graphics processor uses a Weitek XL™ chip set to perform display list traversal, viewing transformations, lighting, and hither clipping operations on each object in the display list. These operations yield a set of colored vertex descriptions[1] for the object which are sent to the frame buffer. Before the graphics processor output is handed to the frame buffer, the image generation controller converts the word-parallel data into the bit-serial format required by the frame-buffer.

The smart frame buffer uses the output of the image generation controller to perform pixel level calculations for area definition, polygon visibility, shadows and shading, sphere/circle formation, and pixel painting. Since pixel level operations are performed on linear expressions of the form Ax+By+C, (where x and y are the image space coordinates of the pixel) a model for the scene in terms of a set of linear expressions is required. The model is developed using the output of the viewing transformation, polygon clipping, and perspective transformation operations performed by the graphics processor. These operations yield a set of screen space polygon vertices. The coordinates for the vertices are used to compute the linear expression coefficients and to develop instructions for the frame buffer. The frame buffer uses the linear expression coefficients and instructions to create the final image display one polygon edge at a time. After the smart frame buffer has formed the image, it is read out by the video controller and displayed.

The formation of an image in the smart frame buffer is accomplished using the logic enhanced memory chips in the buffer. Each chip consists of two 64-pixel modules, as depicted in Figure 4.3. The chip x-address, chip y-address, and y tree multiplier together

[1]Vertex descriptions are sent when processing polygons, individual pixel values are sent when processing medical images.

form the Linear Expression Evaluator (LEE), each of which provides the computing power of two 10-bit multiplier/accumulators at each pixel. The 64 1-bit ALUs and the LEE are used to evaluate the linear expression for the current primitive for all pixels controlled from

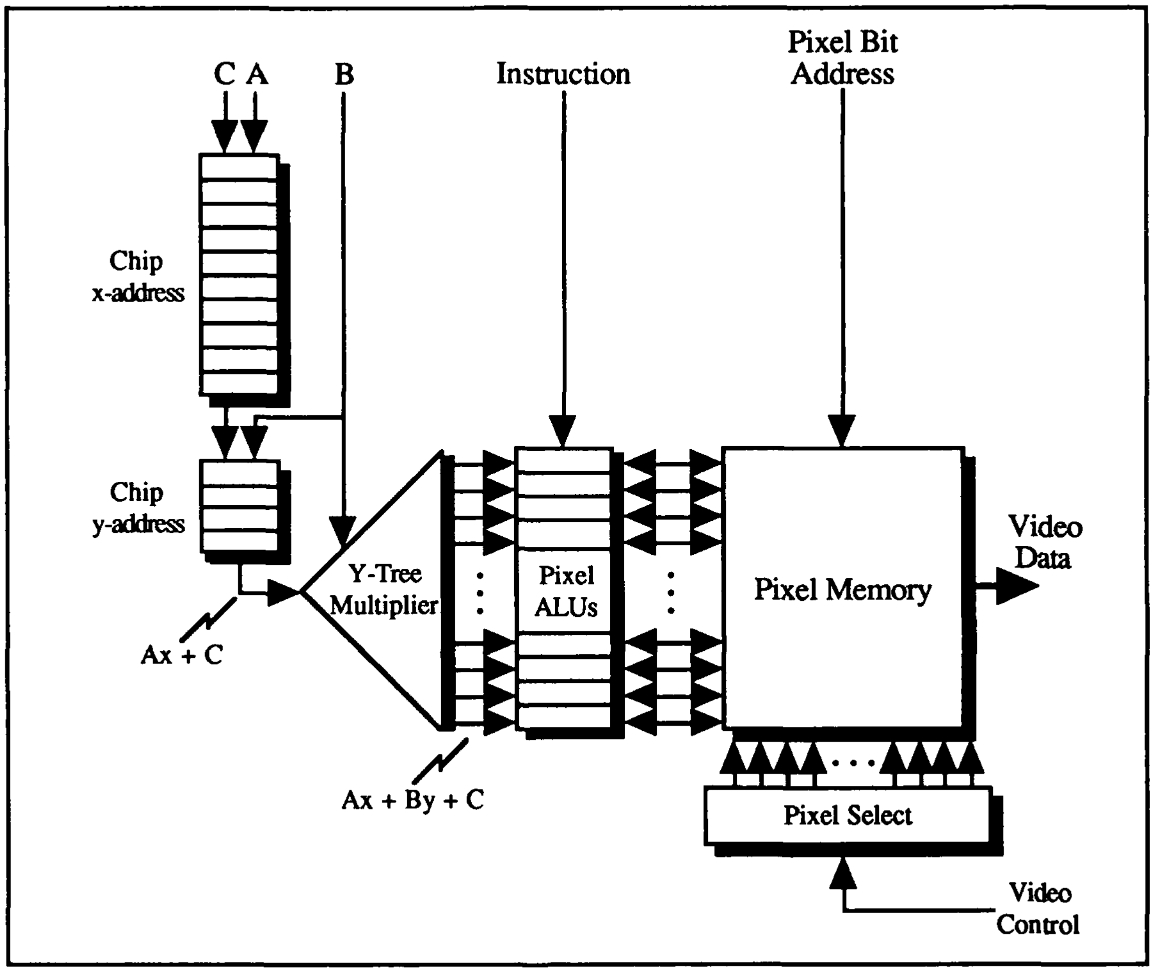


Figure 4.3: Pixel-Planes Logic Enhanced Memory Chip (based upon [Fuc88a]).

the chip. The 72-bit deep pixel memory array is used to store pixel data, such as pixel color, z-depth, texture, and other pixel information, for a 64 pixel column of the screen.

Object display is accomplished in three steps. In the first step, image primitive[1] scan conversion is performed followed by visibility determination of the current primitive relative to previously computed primitives. The final step consists of the operations, such as anti-aliasing, shadows, and shading, required to render the image. Scan conversion is

[1]The Pixel-Planes machines can render polygon, sphere, line, and point primitives.

performed by determining those pixels which lie on or inside a specific convex polygon. This information is computed using the linear expressions provided by the Graphics Processor. Each edge of a polygon is defined by two vertices, $v_1 = (x_1,y_1)$ and $v_2 = (x_2,y_2)$, which are ordered so that the polygon lies to the left of the directed edge v_1v_2. The equation of the edge is given by $Ax+By+C=0$, where $A=y_2- y_1$, $B=x_2- x_1$, and $C=x_1y_2- x_2y_1$. All the coefficients for each edge of a primitive are broadcast in turn. Each LEE evaluates the current expression and passes the result on to the ALUs, which use this information to disable pixels outside the perimeter of the polygon. By convention, vertices are broadcast so that pixels to the right of the directed edge are outside of the polygon. For a given primitive, a pixel lies within the polygon only if enabled after all the edges of the polygon have been broadcast.

Once scan conversion is completed, the visibility of each polygon is determined at the pixel level by a comparison of z-values using the data stored in the pixel memory. The final step, image rendering, includes transparency operations, texturing, shading, shadows[1], and contrast enhancement. The Pixel-Planes machine can employ a unique approach to contrast enhancement within two-dimensional (2D) images[2], termed Adaptive Histogram Equalization, described by Pizer, et. al. in [Piz84]. The motivation for this technique rests on the observation that different pixel values should be assigned different intensity levels so that the viewer can perceive a degree of contrast and detail. But, because the range of voxel values is, in general, greater than the range of intensity levels, some compression of the voxel data range must take place. Common practice is to select a single compression and apply it globally. As a result, control of the contrast in light and dark areas is difficult to achieve. The problem of contrast control can be partially overcome by

[1]Area light source shadows are generated by generating the image multiple times and moving the light source slightly for each pass. A multiple light source capability is under development.

[2]Polygon rendering and Adaptive Histogram Equalization have not been combined in the Pixel-Planes systems.

using Adaptive Histogram Equalization (AHE) to make gray level assignments on a local, not global, level. In this method, the gray level assignment scheme makes use of regional frequency distributions of image intensities when making gray level assignments. This results in greater local contrast between objects, thereby improving image quality. Pizer compares the images obtained using this technique with those developed using threshold based segmentation. The difference in image quality between this technique and simple threshold based segmentation is remarkable; small and fine structures are easily seen while gross anatomical features are clearly differentiated. Because of the length of time required to perform AHE, Pxpl4 first computes a rough approximation of the final version and then progressively refines it. This approach facilitates rapid display of the image during periods of user interaction while permitting the development of higher quality images during static display.

Pixel-Planes 5 builds upon the pixel-level processing foundation laid by Pxpl4, but extends upon it by introducing MIMD processors and a token ring network. These changes, along with a de-coupling of the pixel-processing elements of the "smart" frame buffer from the frame buffer itself and the use of a virtual pixel approach, provide a capability for multiple, simultaneous primitive processing. Additionally, Pxpl5 has a higher clock rate than Pxpl4, has more front-end graphics power with support for host access to individual pixel values, and evaluates quadratic, instead of linear, expressions. The architecture for the Pixel-Planes 5 machine is depicted in Figure 4.4.

The processing strategy used in Pxpl5 is to process primitives one screen patch (128 x 128 pixels) at a time in each Renderer unit, with the use of multiple Renderers to provide a multiple primitive processing capability and inclusion of multiple Graphics Processorsfor assigning primitives to screen patches (bins). Screen patches are

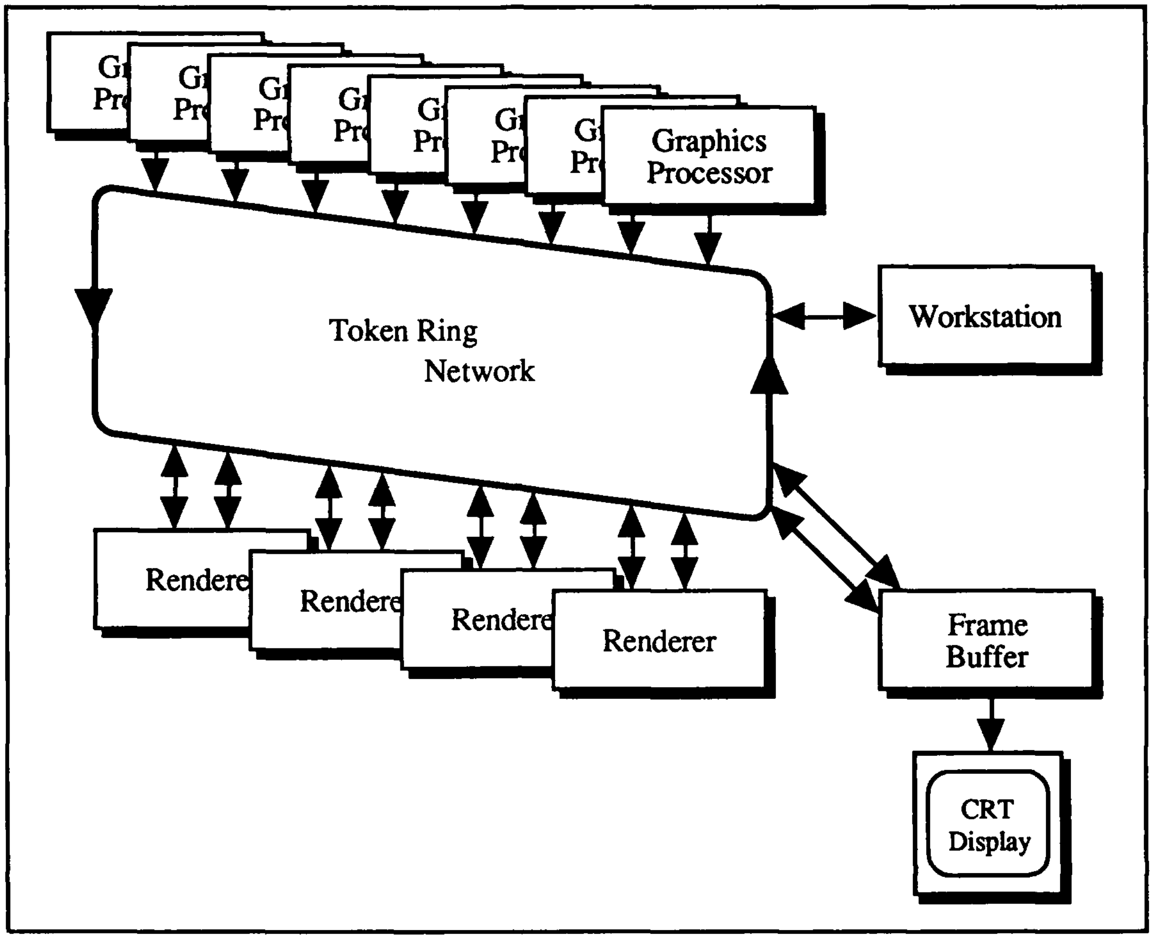


Figure 4.4: Pixel-Planes 5 Architecture (based upon [Fuc89b]).

dynamically assigned to Renderers during processing, with each Graphics Processor sending its primitives in the patch to the appropriate Renderer in turn. This strategy is realized in the architecture depicted in Figure 4.4. The host workstation is responsible for user interaction support and image database editing. The 32 Graphics Processors (GP) are individual Multiple-Instruction, Multiple-Data (MIMD) units running identical C™ code employed to handle the Pixel PHIGS (a variant of PHIGS+[1] designed for Pixel-Planes) data structures and to sort its set of primitive objects into bins corresponding to parts of the screen. Each of the 8 - 10 Renderers in the system, depicted in Figure 4.5, are

[1]PHIGS+ (Programmers' Hierarchical Interactive Graphics System) is an extension to the ANSI standard's graphics committee current PHIGS 3D graphics standard which supports lighting, shading, complex primitives, and primitive attributes. A PHIGS tutorial can be found in [Cah86].

independent Single-Instruction, Single-Data (SIMD) processing units descended from the Pxpl4 memory processing chips, which together operate on a single 128 x 128 patch of the screen at a time. Additional memory on the board serves as a backing store used to retain pixel color values for the patches of the screen processed by the Renderer. Each Renderer chip has 256

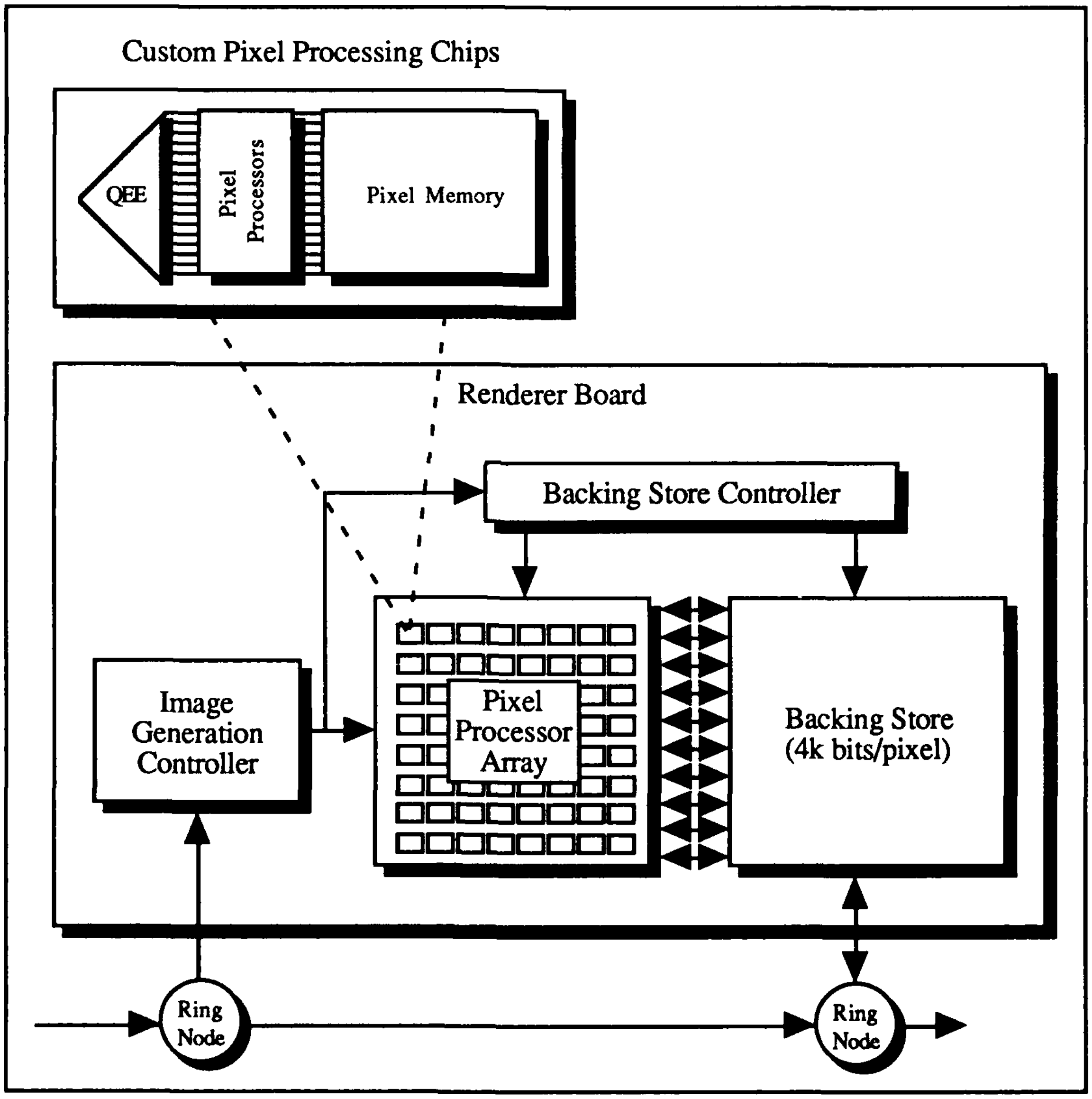


Figure 4.5: Pixel-Planes 5 Renderer (based upon [Fuc89a]).

pixel processing elements and 208 bits of static memory per pixel, which are used to hold color, z-depth, and other associated pixel information. The quadratic expression evaluator on the chip is used in much the same way as the LEE in Pxpl4, except that it can evaluate the quadratic expression $Ax + By + C + Dx^2 + Exy + Fy^2$ at each pixel using the A,B,C,D,E, and F coefficients broadcast from the GP. The 1280 x 1024 frame buffer is double-buffered in order to meet the 24 frames/second screen update rate performance target performance of Pxpl5. The individual units of the system are interconnected using a multi-channel token ring network which can support up to eight simultaneous messages at 20 Mwords/second.

Rendering of a new image is initiated when an application on the host workstation edits the image database and sends these changes to the Graphics Processors via the network. One Graphics Processor (GP) is designated as the Master GP and has the responsibility for assigning Renderers to portions of the screen and informing the GPs of the Renderer assignments. Using the changes sent from the host, each Graphics Processor transforms the primitives it is responsible for and then sorts them into the correct bins according to the virtual screen patches the primitives overlay. Once sorting is complete, the Master GP makes the initial Renderer assignments and informs the GPs of the assignment. A screen patch is rendered by each GP broadcasting, in turn, the bin containing the primitives and instructions for processing them to the Renderer with the corresponding screen patch assignment. Once a GP has completed its broadcast to a Renderer, it notifies the next GP that it may begin processing, and then waits for its next turn to broadcast. The final GP to broadcast its bin to the Renderer informs it that the current screen patch at the Renderer is complete. The Renderer then moves the pixel color values to the backing store and is assigned a new patch by the Master GP. When the entire image has been rendered, each patch is transferred to the frame buffer to refresh the display.

Pixel-Planes 5 can perform the same operations as Pxpl4, but the use of quadratic expression evaluators and its higher rendering speed provide additional capabilities. For medical imaging, volume rendering can be accomplished by storing the voxels in the scene within the backing store for each Renderer. The algorithm used to render the image begins with a 3D array of voxels which are individually classified and shaded at the Renderers using the Gouraud shading model. The resulting color and opacity values for each voxel are retained in the backing store of the Renderer. Parallel viewing rays are then traced into the array from the viewer's position using the GPs, with the Renderers transmitting requested voxel values to the GPs for trilinear interpolation of the eight closest neighbor voxel values and composition with the current pixel value in the GP. The resulting pixel value is sent to the frame buffer for display. Pxpl5 is expected to achieve between 1 and 10 frames per second performance depending upon desired image quality. In addition to the volume rendering operations required for medical imaging, Constructive Solid Geometry defined objects can be rendered and faster image texturing and transparency effects can be achieved in Pxpl5 than in Pxpl4.

4.3.2.1 Shading, Anti-aliasing, and Hidden-Surface Removal Techniques.

The input data must be convex polygons[1] or voxels. Shading is accomplished by broadcasting three sets of coefficients for each linear expression, one for each primary color component. Gouraud and modified Phong shading can also be used by linearly interpolating the colors at the vertices of the polygon. Pixel-Planes 5 can employ the Phong shading model directly and most likely will use a radiosity lighting model[2] to perform shadow calculations rather than using the plane equations for shadow volumes as in Pxpl4. The standard z-buffer algorithm is used for hidden-surface removal. High speed is obtained by simultaneously applying the z-buffer algorithm to each pixel.

Pixel-Planes uses a form of supersampling for anti-aliasing. Each pixel is subdivided into a grid of subpixels so that each subpixel has an address of the form x+offset, y+offset from the center of the pixel. The image is generated several times

---

[1]A convex polygon is shaped so that any two points within the polygon can be connected by a line which does not pass through the perimeter of the polygon, see [Fol83].

[2]The radiosity lighting model accounts for diffuse interreflection between surfaces within an image.

(Fuchs reports using 16 times) in such a way that the sample points within the area of a pixel form a reasonable distribution. Two sets of color buffers are maintained, one stores the color generated by the latest image generation offset, the other stores a running average. The display is formed using the values in the running average buffer after all offsets have been processed. The technique is similar to the local average technique described by Crow (see [Cro77b], [Cro81]).

4.3.2.2 Degree of Parallelism and Machine Performance.

Pixel-Planes 4 is an SIMD machine wherein all processors in the "smart" frame-buffer execute the same instruction simultaneously at the pixel level with an output up to 30,000 polygons per second. However, Goldwasser, in [Gol85a], reports that a typical medical image contains 500,000 polygons per frame. Based on this figure, Pixel-Planes 4 is not capable of real-time performance in the medical imaging environment, as it requires approximately 20 seconds to generate a single frame. Pixel-Planes 5 is a hybrid architecture, with SIMD and MIMD components, with a predicted capability for rendering 256 x 256 x 256 medical images at the rate of 1 - 10 frames/second.

4.3.3 Kaufman's Cube Architecture.

Kaufman's Cube machine, see [Bak89], [Coh90], [Kau85], [Kau86], [Kau87], [Kau88a], [Kau88b], [Kau88c], [Kau89], and [Kau90], was developed to permit 3D interaction with medical images and geometric objects in real-time. The system development plan calls for the production of a system with 512 x 512 resolution of a 512 x 512 x 512 voxel scene, though the hardware prototype supports only a 16 x 16 x 16 voxel scene. A diagram of the system, based upon [Kau88b] is presented in Figure 4.6.

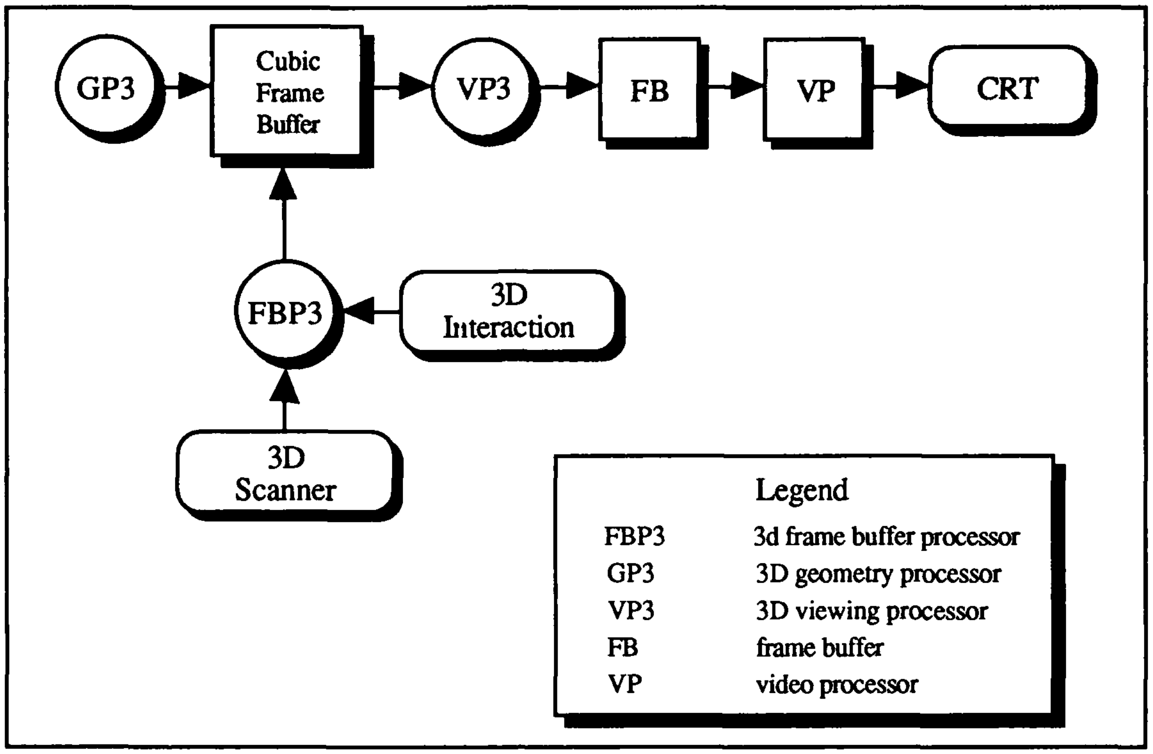


Figure 4.6: Kaufman's Cube Machine Architecture.

The processing strategy for the machine consists of providing a suite of medical image processing capabilities that support the generation of shaded projections of 3D voxel data with the provision for colorizing the output and selection of object(s) to be displayed in a translucent manner. The processing strategy is supported through the use of parallel processing to simultaneously process an entire beam[1] of voxels for viewing while permitting user interaction with the system in 3D space through the host. Color selection, translucency control, and slicing can be altered by the user in real-time, thereby allowing for the inspection of interior characteristics of the volume. The design goal for the machine is to provide these capabilities in real-time within its own workstation environment. A unique approach is taken within the machine in its approach to performing image transformations. Shifting, scaling, and rotation are performed by calculating the new coordinate for a voxel based upon the new coordinate position taken by the closest

[1]A beam of voxels is a row, column, or axle of voxels within the scene.

neighbor voxel previously transformed. This approach reduces the number of arithmetic operations performed to no more than three additions per voxel [Kau89].

The image processing system consists of a host and four major components, the 3D frame buffer, the cubic frame buffer, 3D geometry processor, and the 3D viewing processor which includes the voxel multiple write bus (VMWB). The host is responsible for the user interface. The 3D frame buffer processor (FBP3) is responsible for loading 3D voxel-based data of icons, objects, and medical images as well as for manipulating the data within the cubic frame buffer (CFB) when arbitrary projections of the data therein are required. The geometry processor (GP3) is responsible for scan conversion of 3D geometric models into voxel representations for storage within the CFB. For a description of the geometric scan conversion process see [Kau86], [Kau87], and [Kau88a]. The (CFB) is a large, 3D, cubic memory with sufficient capacity to hold the entire 512 x 512 x 512 x 8-bit scene at once. To facilitate rapid retrieval and storage of an entire beam of voxels, the CFB uses a skewed memory scheme. The memory is organized into n memory modules such that, for any desired orientation, no more than one voxel will be required from each module. The voxels are assigned to memory modules according to their (x,y,z) object space coordinates, with all of the voxels in module k satisfying the relation k = (x + y + z) mod n. The internal mapping within the module is accomplished using the x and y coordinates of the voxel. When forming an image, voxels are retrieved beam by beam and placed in the VMWB for selection of the voxel in the beam to be displayed.

The 3D viewing processor (VP3) fetches beams of voxels from the image stored in the CFB and uses the VMWB to extract orthographic projections along the specified viewing axis. The VP3 also performs slicing, clipping, depth determination, shading, and color determination on each voxel before sending this data to the frame buffer, which displays the shaded orthographic projection of the scene within the CFB. To meet the requirement for real-time image generation, all the voxels in the beam are processed simultaneously by the VP3 using the voxel multiple write bus. The voxel selection algorithm employed in the VMWB enables the VMWB to select the voxel closest to the observer in log time. The VMWB has n processors for a scene n voxels deep, or 512 processors for the CFB example cited previously. To process the scene, 512^2 sequential accesses to the CFB by the VMWB are made in order to to select the beams of voxels in the scene for display. The VMWB determines the visible voxel in each beam and sends the density value for the voxel to the 3D viewing processor.

The key elements to the ability of the Cube architecture to rapidly form an orthographic projection are the use of a unique memory configuration in the CFB and the VMWB design. The CFB memory organization supports simultaneous access to a beam of voxels and the design of the VMWB enables rapid determination of the opaque voxel in the beam closest to the observer. If an arbitrary rotation of the scene in the CFB is called for, a two-step procedure is performed. First, the FBP3 extracts the scene and rotates it using the incremental transformation technique. The VP3 then accomplishes its normal procedure for displaying a view of the scene along an orthographic direction.

An enhanced linear skewing scheme for CFB address allow for rays along non-orthographic parallel projections to be retrieved conflict-free. The viewing architecture has been extended, with the addition of three additional 2D buffers, to accommodate arbitrary parallel and perspective projections. Projections in this architecture are performed by retrieving a plane of projection rays from the CFB and then employing the additional buffers to align the plane for conflict-free projection-ray retrieval. Every projection ray within each plane is then analyzed by the VMWB to determine the projection of that ray. Additional information on this enhancement is to appear in [Kau90].

4.3.3.1 Shading, Anti-aliasing, and Hidden-Surface Removal Techniques.

The voxel data model is used throughout the machine, so image data preprocessing is not required. The Cube machine also has the capability to manipulate voxel-based image overlays at the same time as the medical image is formed. This capability supports the use of icons, cursors, synthetic needles and scalpels, and models of parts of the body, such as a skull, to aid in diagnosis of disease and treatment planning, as discussed in [Kau86] and [Bak89]. Shading is accomplished using the congradient shading method, described in [Coh90]. This method defines the surface orientation of a voxel face using a set of neighborhood codes, much like the normal-based contextual shading technique described earlier. However, the neighbor codes are determined by computing the gradient between the voxel face and the faces of the adjoint voxel faces, as in gradient shading. This method defines the gradient as one of a finite set, allowing the use of look-up tables for gradient determination and also permits changing light source parameters by redefining the look-up tables. The congradient algorithm is implemented as part of the Cube image rendering pipeline, allowing for the production of shaded images in real-time. Anti-aliasing is not performed.

Hidden-surface removal is accomplished using a front-to-back algorithm implemented implicitly in the VMWB, as the VMWB selects the voxel closest to the observer for each beam. The voxel selection algorithm uses the location of each voxel along the beam and its density value to determine which voxel in the beam is visible. To avoid calculation of the location of each voxel on the beam, each processor in the VMWB is assigned an index value which is stored in each unit, and sequentially numbered so that the lowest index value lies at the back of the beam. The index value for each processor is treated as the depth coordinate of the voxel it contains when voxel selection is performed. When a beam of voxels is processed, the location of the clipping plane and any color(s) considered to be transparent are broadcast to all the processors on the VMWB. If a processor contains a voxel that is transparent or lies either in front of or behind the clipping plane, the processor disables itself during the current round of beam processing. Each of the remaining processors then start placing its index value on the Voxel Depth Bus bit by bit starting with the most significant bit. As each bit is written, each processor examines the bit value written to the bus, and if it not equal to the bit value the processor just wrote, the processor disables itself for the duration of the processing of the current beam. One processor eventually remains, and the voxel data and the depth at this processor is then sent to the VP3 for pixel colorization and shading .

4.3.3.2 Degree of Parallelism and Machine Performance.

The Cube machine exploits parallelism at the beam level, where it processes full beams simultaneously using the CFB skewed memory organization and the simple logic in the VMWB. The estimated machine performance varies depending on the type of operation it must perform. Orthographic projection, shading, translucency and hidden-surface removal of the scene in the CFB can be accomplished in .062 seconds for a 512 x 512 scene, which is roughly real-time. However, if an arbitrary projection of the 512^3 imaged volume is required, 0.52 seconds are required to form the 512^2 image. Use of the enhanced linear skewing scheme results in an estimated rendering time of .16 seconds. These estimated performance figures are based upon printed circuit board technology, higher speeds are anticipated with the VLSI implementation currently under construction.

4.3.4 Medical Image Processing Group Machines.

The Medical Image Processing Group(MIPG) has developed a series of machines (see [Art79a], [Art79b], [Art81], [Che84a], [Che84b], [Che85], [Edh86], [Fri85a], [Fri85b], [Her78], [Her79], [Her80a], [Her82], [Her83], [Her85a], [Her86], and [Rey83b]) that produce 3D displays of medical images using surface shading and hidden-surface removal to impart a sense of 3D relationships on a 2D display. The following material describes the operation of 3D98, the latest of the machines. This machine is unique in that the primary design objectives are low cost with acceptable image quality rather than rapid image formation. For this reason, surface tracking and contour extraction techniques were developed to reduce the volume of data to be manipulated.

3D98 employs whatever microprocessor controls the Computed Tomography (CT) scanner to perform all image processing calculations. The processing strategy has two components. First, reduce the computational burden. This is accomplished using image segmentation techniques to isolate the object of interest in the 3D volume early in the image formation process and by employing a binary array to represent the isolated object rather than the entire 3D volume. The second portion of the strategy further reduces the computational burden by representing the surface of the object with a list of cuberille faces instead of the binary array. The result is that the amount of data to be processed for hidden-surface removal and shading is greatly reduced from that provided by the CT. However, much scene information is lost along the way, and any change in the object of interest requires that the image be reprocessed.

Image processing proceeds through four steps. The first two steps produce a binary array of cuberilles from a sequence of cross-sectional voxel-based slices. The first step interpolates the values in the CT scanner slices. The second step consists of selecting the object of interest by thresholding with the resulting image stored in the binary array. The entries in the array which contain ones correspond to cuberilles within the object of interest. The entries in the array which contain zeroes correspond to cuberilles in the scene background. The third step is surface tracking. The input to this step is the binary array produced in step one, the output is a list of faces of cuberilles. Each face on the list lies on the border of the object and so separates a voxel labeled zero from a voxel labeled one. The faces on the list form a closed connected surface containing the object which contains the cuberille face identified by the user as the starting point for the surface formation process. Further details can be found in [Her83] and [Fri85b]. The fourth and final image processing step is shaded surface display. The input to this step is the surface defined in the list output from step two. The output is an image showing the appearance of the medical object when viewed from a user-specified direction.

4.3.4.1 Shading, Anti-aliasing, and Hidden-Surface Removal Techniques.

The MIPG machines employ the cuberille model. The cuberilles are formed by interpolating the CT scan voxel data. 3D98 uses normal-based contextual shading, a shading algorithm which estimates the surface normal for the center of each visible cuberille based on the orientation of the cuberille orientation and the orientation of the eight neighboring cuberilles. This technique is described in [Che84a] and [Che85]. No anti-aliasing is performed as this would serve to increase the computational burden.

A modified z-buffer algorithm is used for hidden-surface removal. Because of unique properties of the cuberille environment, Herman and Liu [Her79] were able to demonstrate that a point in a face can be hidden by a point in another face only if the center of the first face is closer to the observer than the center of the second face. This property eliminates the need for computing the distances to individual points in the faces, and only distances to the centers of faces need be computed when using the z-buffer algorithm for hidden-surface removal.

4.3.4.2 Degree of Parallelism and Machine Performance.

The machine produces images slowly. A single 256 x 256 image can take 1-3 minutes to display. However, high performance is not an objective of this machine. There is no parallelism in the machine. There is one CPU, that found in the CT scanner, and it performs all the calculations. For actual clinical use the MIPG machine is used to precompute selected display sequences during offpeak CT scanner use periods for later viewing.

4.3.5 Reynolds and Goldwasser's Voxel Processor Architecture.

The Voxel Processor architecture was proposed and described by Reynolds and Goldwasser in [Gol83], [Gol84a], [Gol84b], [Gol85a], [Gol85b], [Gol87], [Gol88b], [Rey83a], and [Rey85]. The Voxel Processor machine processes discrete subcubes of an image in parallel and outputs the results of these computations to a raster scan display. The Voxel Processor is a special purpose machine whose only application is the parallel processing of cuberille based medical images[1].

The Voxel Processor machine seeks to provide the capability for volume matting, density window selection, rotation, translation, scaling, and surgical procedure simulation operations on a 3D volume to generate a 2D shaded display in a real-time[2]. The processing strategy divides the scene into small volumes composed of cube-shaped voxels which are independently processed and then merge the resulting small images in back-to-front order to form one final image. A key element of the processing strategy is that, in order to attain real time performance, the algorithms to accomplish merging and small image generation are placed in hardware instead of software. Additionally, no operation more time-consuming than an add, shift, or comparison is used to generate the final image. There are seven components of the image processing pipeline. These components are the host computer, the object access unit, the object memory system (64 modules each storing a 64 cube) , the processing elements (PE's), the intermediate processors (IP's), the output

[1]A commercial machine incorporating the parallel primitives introduced in the Voxel Processor architecture is described in [Gol88a].

[2]Real-time display requires the display of 25 - 30 frames/second. The Voxel Processor achieves a frame rate of 25 frames/second.

processor (OP), and the post processor. A diagram of the machine, based upon [Gol87], appears in Figure 4.7.

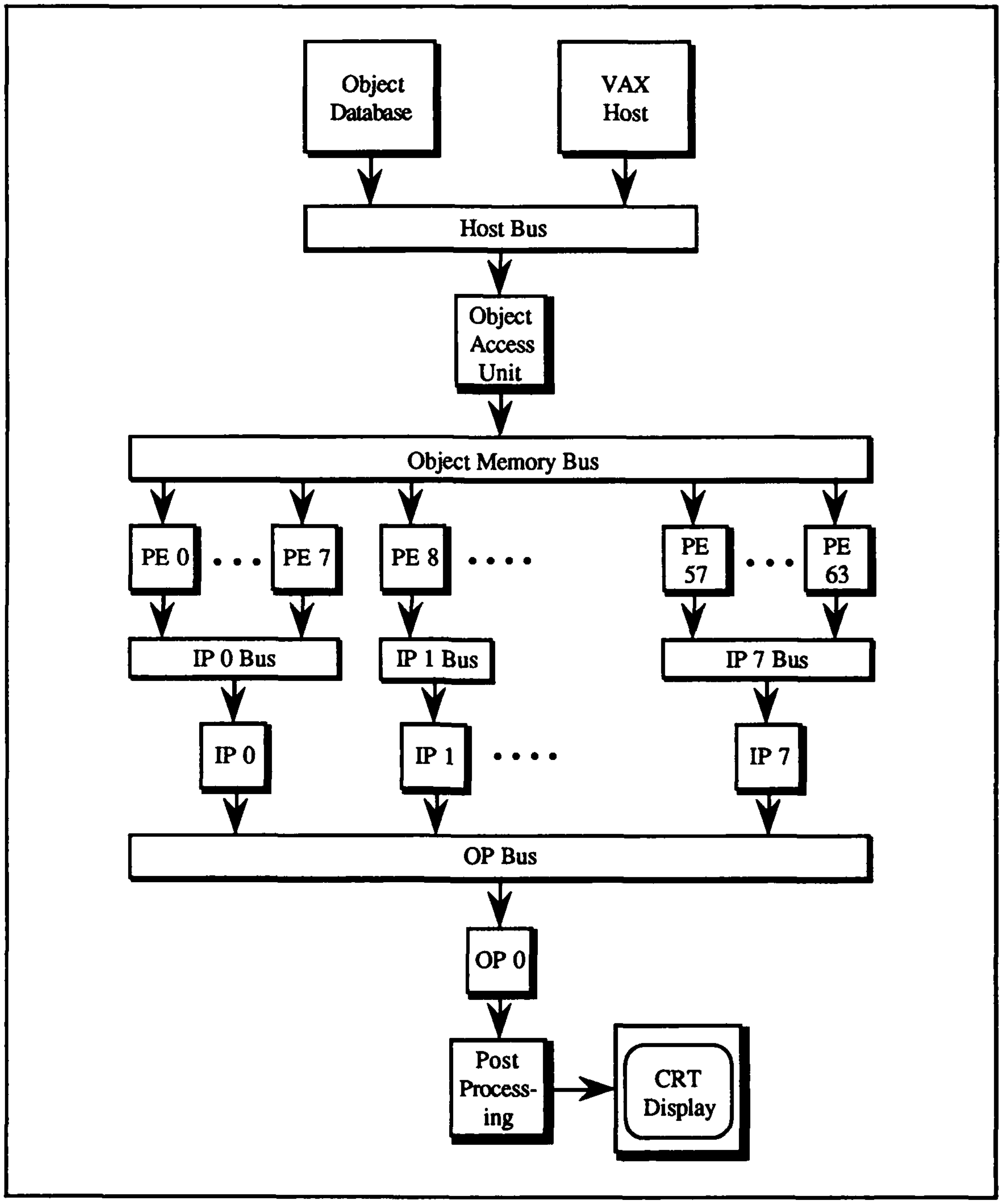


Figure 4.7: The Voxel Processor Architecture (based upon [Gol87]).

The host computer handles object data acquisition, database management, and complex object manipulation. The host is also responsible for generating the Sequence

Control Tables (SCTs) which are used to control the back-to-front object readout sequence. The object access unit supports object database management by the host and manages communication between the Voxel Processor and host. The object memory system (OMS) provides the 16M bytes of RAM required to hold the 256 x 256 x 256 image. The OMS stores the voxel-based object data within 64 memory modules distributed among the processing elements. The processing elements are used to form mini-pictures from their 64 x 64 x 64 cuberille sections of the scene. Each PE has two 128 x 128 output buffers, a copy of the Sequence Control tables, an input density look-up table, and a pipelined arithmetic processor. During operation, the PEs access the data in its own subcube in back-to-front order without contention with other PEs. This computation yields the 2D sub-image that is visible given the current set of user editing inputs and object orientation. This image is composed of visible voxel density values and their image space z-distance values. The 2D image is placed into one of the two output buffers[1] at the PE for use by the IP. The next two stages of the pipeline perform the task of merging the 64 separate pictures into a single picture that portrays the volume. The merge operations are performed using the SCT to determine the input picture position offsets and memory addresses. Each of the eight IPs merge the mini-pictures generated by its set of eight PEs into one of its two 256 x 256 output buffers. The OP forms the final image by merging the contents of the eight IP output buffers into the 512 x 512 frame buffer. Once the image is in the 512 x 512 frame buffer, post-processing is performed by the post-processor. The post processor is responsible for shading, brightness, and pseudo-coloring of the final image using texture maps and either distance shading or gradient shading.

Two critical steps in the operation of the machine are subimage merging at the IPs and OP and mapping the voxels from object space to image space at the PEs. These operations are accomplished under the control of two Sequence Control Tables (SCTs).

---

[1]The 128 x 128 output buffers can hold the largest image that can be created from a 64-cube.

Each of the two SCTs is computed by the host computer based on the current orientation of the object and then broadcast to the processors in the machine. SCT1 contains the back-to-front readout sequence of the scene and is used by the IPs and OP to control the merging operation performed on the output of the PEs and IPs. SCT2 is used by the PEs to map the voxels within the PEs from object space to image space. Each table contains eight entries, one for each octant of a cube, with the entries in each table sorted in back-to-front order to support the back-to-front recursive rendering of each sub-image in the PEs and back-to-front merging of sub-images in the IPs and OP.

4.3.5.1 Shading, Anti-aliasing, and Hidden-Surface Removal Techniques.

Image shading is accomplished using gradient shading or distance shading. Distance shading is accomplished by reducing the intensity of a pixel depending upon its distance from the observer. Gradient shading is a more comprehensive shading algorithm in that it takes observer distance and local surface curvature into account when shading a pixel. The curvature is approximated by determining the z-gradient between a pixel and its 6 neighbors. The gradient value is used to approximate the surface normal at the pixel. The pixel is then assigned a shade based upon the color of the light source, the angle between the surface normal and the incoming light, and the distance of the pixel from the observer. Reynolds and Goldwasser report that they have investigated two different anti-aliasing techniques. One method is the display of the centers of the visible faces of each voxel, which works only if the size of a voxel face is the same size as a pixel (see [Art81] and [Her79] for details). The other method is supersampling to double resolution. Goldwasser and Reynolds claim that both these anti-aliasing techniques yield acceptable results for medical imaging.

Hidden-surface removal is accomplished using the recursive back-to-front algorithm described in [Rey85] and [Gol87]. Briefly, the algorithm operates as follows. The entire volume is conceptually divided into octants of dimension 2^n x 2^n x 2^n, and the x, y, and z object space coordinate for each octant center is determined. To achieve a given object orientation, rotation matrices (see [Fol83]) are applied to these eight sets of coordinates, yielding the image space coordinates for the octant centers. The values for the image space coordinates are left-shifted to maintain precision during subsequent computation and sorted into decreasing z-value order (yielding the SCTs mentioned previously). The image is rendered by recursively computing the coordinate values of the centers of successively smaller subcubes in image space in concert with their corresponding subcube centers in object space. Eight new subcube center coordinate values are then computed in back-to-front order. This computation proceeds by accessing the SCT entries in decreasing z-value order, shifting each SCT coordinate entry, and its corresponding object space coordinate value, an amount determined by the depth of the recursion, and adding this result to the coordinate value of the parent subcube. At recursive level n, the coordinate values of individual voxels in object space and image space are obtained. These coordinate values, after right-shifting, can then be used to retrieve the density value for each voxel from an object array and store it in an image space array.

4.3.5.2 Degree of Parallelism and Machine Performance.

The Voxel Processor employs parallelism at two different levels. Within each of the first two stages of the processor pipeline, there is parallel operation on disjoint sets of voxels. This allows the processing components of each stage to operate independently of all other processors in its stage. Second, each stage of the pipeline operates independently due to the dual-buffering of the output from each stage. Therefore, each stage can independently operate on a different output frame, so that at any one time there can be four frames in the pipeline. This high degree of parallelism is re nsible for the real-time performance, twenty-five 512 x 512 frames/second, attained by the Voxel Processor.

4.3.6 Robb's True 3D Machine.

Robb's machine, [Harr79], [Harr86], [Hef85a], [Hef85b], [Rob85a], [Rob86a], [Rob87], and [Rob88], is the only machine in this survey which provides a true 3D display. The image is formed without employing parallel processing with the image generation operations accomplished in a workstation. The design objectives of the system are directed toward providing an environment suitable for rapid visualization and analysis of volumes of 3D data. Specifically, the architecture is geared to display up to 500,000 voxels/second as a continuous true 3D image. The system is required to support numerical tissue dissolution and image dissection in order to isolate objects of interest within the volume. Finally, the system operator must be able to alter many of the display parameters as the image is being displayed. The central concept of the system is that 3D image analysis is an editing task in that proper formatting (editing) of the scene data will highlight those 3D relationships that are important to the user. In addition, the system supports a wide variety of interactive 2D scene editing, display, and analysis options which are used to characterize the data set.

The image processing machine is a Charles River Data Systems Universe system using a Motorola™ 68020 running at 12.5 Mhz. The workstation has 10MBytes of memory and two video display terminals with 512 x 512 resolution which support 3D scene editing operations. In addition, a varifocal mirror assembly consisting of a mirror, a loudspeaker and CRT[1], is connected to the system to provide the true 3D images. A diagram of the system, based upon [Rob85], is presented in Figure 4.8.

[1]The loudspeaker is used to vibrate the mirror synchronously with the CRT display of 30 frames/second.

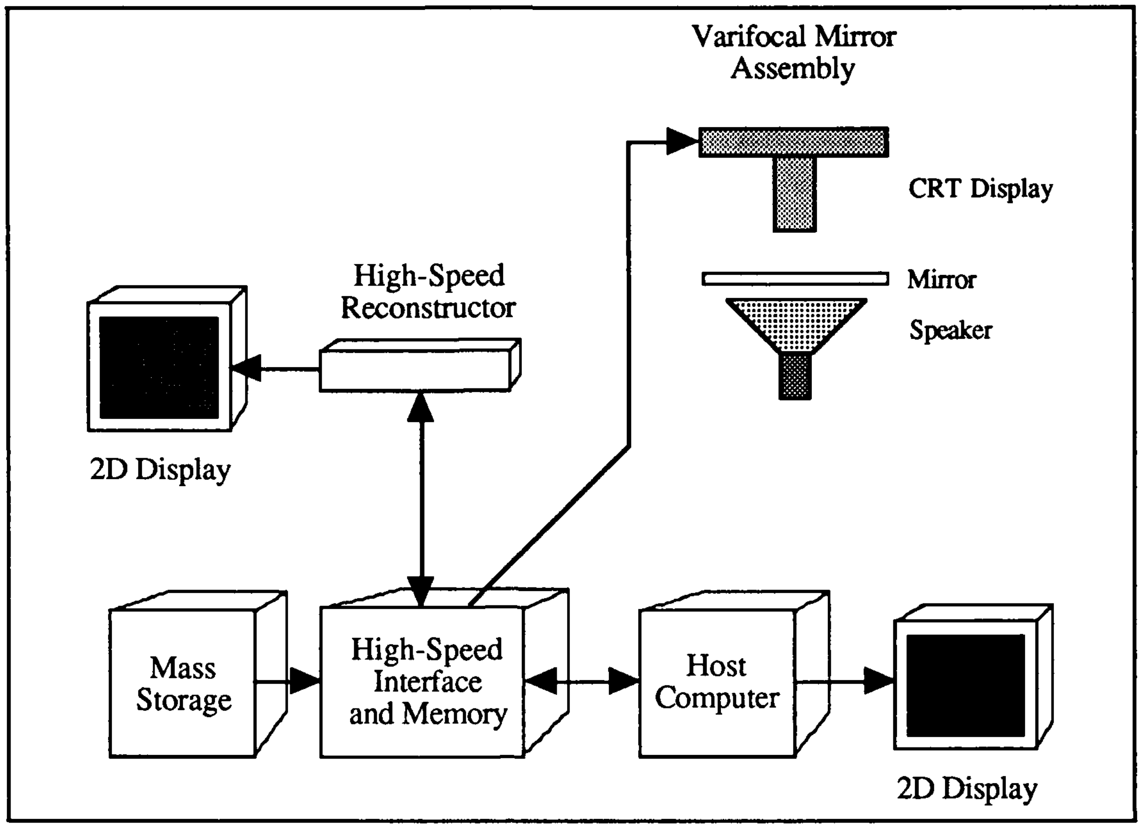


Figure 4.8: True 3D Machine Architecture.

The system hardware and data format are designed to permit rapid image production. The display of the image is accomplished by storing the data slices in a high speed 2D memory from where they are output at a real-time rat to a CRT which projects onto a varifocal mirror. The key to the high-speed display is the use of a custom video pipeline processor to perform intensity transformations on the voxel data before display. The intensity transformation is controlled by a two-bit field appended to the density information for the voxel. Upon entry to the pipeline, the field is stripped from the remainder of the data and examined. The bit settings determine whether the voxel is given a maximum intensity value, set to black, passed through unchanged, or modified by a table look-up. Once the pipeline has completed processing, the transformed voxel is sent to the CRT for display upon the varifocal mirror.

Within the True 3D system, image processing is performed by a set of communicating processes running within the workstation, with each process designed to perform a different class of operations. The processes implement the graphics functions used for display on the 2D devices, user interface management, interprocess communication, and for 3D object editing tasks and have been grouped together into a system named ANALYZE[1]. To facilitate rapid processing of the image by the various editing processes, the entire image is maintained as a shared block of memory. The operation of these modules is discussed fully in [Rob86a] and [Rob88] and is summarized here. The hierarchical relationships between the ANALYZE processes are portrayed in Figure 4.9.

Image processing commences when the desired image volume is called into the machine by the host workstation using the TAPE or DISK process and the selected volume of data is converted to the format required by the 3D display. The DISPLAY process performs manipulation and display of selected 2D sections within the 3D volume as well as 3D transmission and reflection displays of the entire data set. The 2D section display process is used to produce images of slices in the 3D data set which lie parallel to one of the three coordinate axes. The slices to be extracted are defined by a viewpoint which can lie on one of the major coordinate axes. The process uses this information to generate an arbitrary number of sequential 2D views of the dataset as seen from the viewpoint. The 2D section display process supports thresholding, smoothing, boundary detection, and 90° rotation to produce the desired 2D views.

The three-dimensional transmission and reflection displays are formed using ray tracing techniques to render parallel projections of the 3D data set from arbitrary viewpoints. A transmission display, which forms an image akin to an x-ray, can be formed from either the brightest voxel along each ray or the weighted average of the voxels along each ray which lie within a specified threshold. A reflection display, which provides

[1]This software package also runs on standard UNIX workstations such as those from Sun Microsystems, Inc. and Silicon Graphics, Inc and is available, minus the Mirage module, from the Biodynamics Research Unit at the Mayo Clinic.

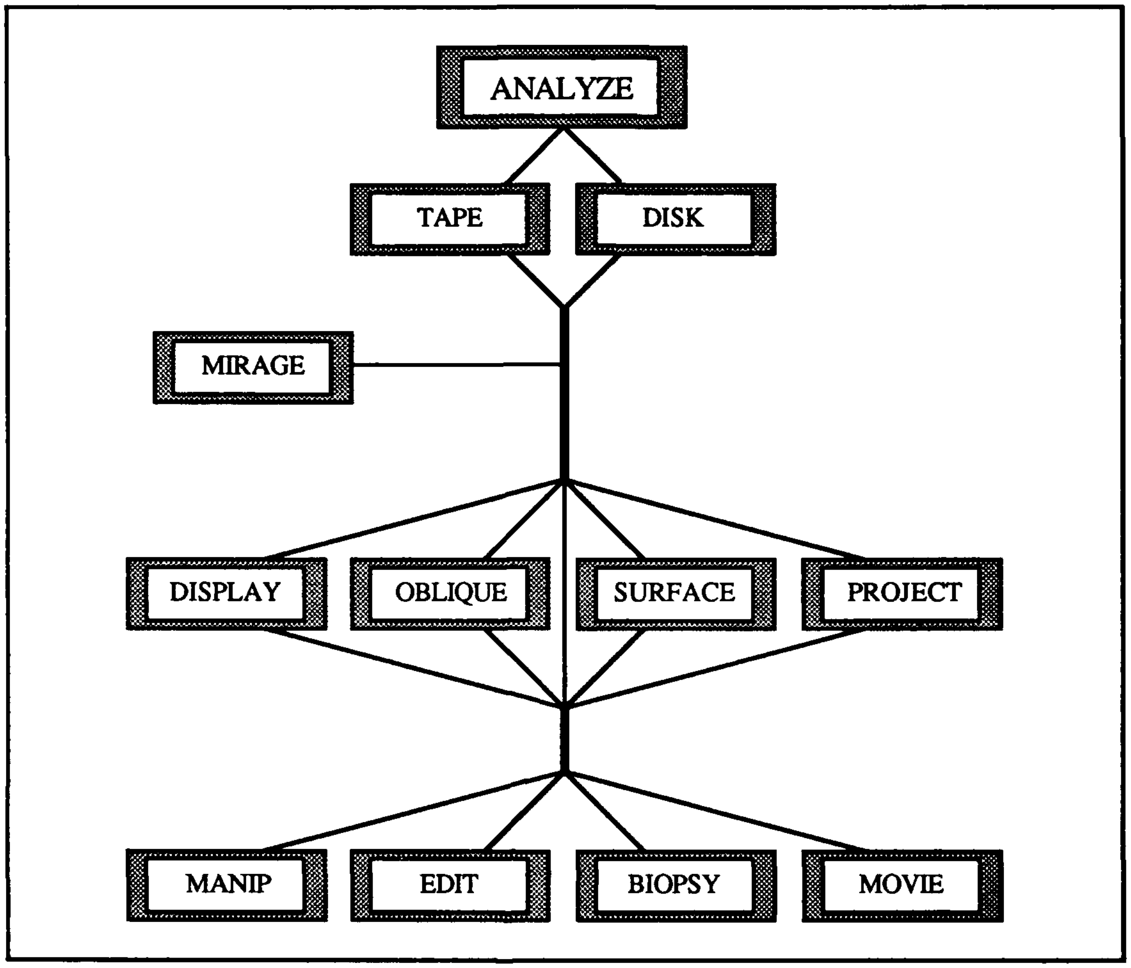


Figure 4.9: ANALYZE Processes in the True 3D Machine (from [Rob86a]).

an image closely resembling a photograph, can be used to form 3D shaded-surface displays or transparency displays. Shaded-surface displays are formed via termination of the ray whenever a voxel lying within a specified threshold is intersected by the ray, which essentially is a technique for 3D surface extraction by thresholding. Transmission displays are generated using an α channel image composition of a transparent and underlying opaque surface within the volume. The value returned for each ray is calculated from the lighting intensity values calculated for the reflection due to the transparent surface, the light transmitted through the surface, and the reflection from the opaque surface weighted by a calculated α channel value. The calculated α channel value is based upon the transparent surface's orientation to the light source, a weighting value for α based upon the orientation, and an assigned maximum and minimum α channel range for the transparent surface[1]. Note that only an opaque surface and a transparent surface are extracted from the volume to form the image, additional surfaces are inhibited from appearing in the final image by thresholding.

The OBLIQUE process generates and displays oblique views of the 3D volume on the 2D CRT. OBLIQUE performs many of the same 2D-slice display functions as DISPLAY, but is not limited to forming projections of the volume along one of the major axes. The volume is represented on the CRT by a cube shape, with the current oblique cutting plane displayed within the cube. As the operator manipulates the orientation of the cutting plane, the system responds (within one second) with a 2D display of the desired section. The computations in this process are performed using a nearest neighbor interpolation to facilitate rapid formation of the 2D display.

The MIRAGE process is the heart of the True 3D display system. This process consists of several modules used to format the data, to perform arbitrary rotations of the formatted data, to window-out selected voxel values, to specify and display oblique planes through the data, and to accomplish data dissolution/dissection[2]. The results of these computations are displayed upon the varifocal mirror as a true 3D image behind the mirror. Data formatting is a pre-processing operation which transforms the 128 input image planes down to 27 display frames and appends the two-bit fields to the voxel values. These 27 remaining frames are used to generate the 3D image and can be interactively edited by other modules within MIRAGE. The two-bit display field discussed above is used by MIRAGE for dissolution, dissection, and windowing operations. Dissection is accomplished by

---

[1]The formula, from [Rob88], is: $I = I_{tsp} * \alpha + (1-\alpha) * I_{opq}$ where $\alpha = \alpha_{min} + (\alpha_{max} - \alpha_{min}) * \cos p\theta$. α_{min} and α_{max} define the light transmission coefficient range, θ is the angle between the surface normal and the light source, and p is the distribution coefficient for the α channel range which is assigned according to $\cos\theta$, p normally lies in the range $1 \leq p \leq 4$.

[2]Dissolution appears as tissue dissolving, or melting, away. Dissection appears more as a peeling away of overlying layers.

using MIRAGE to set the control fields of the region to be removed to the black setting for the pipeline. Dissolution is accomplished in a similar manner, except that voxels are gradually faded to black using the look-up table in the pipeline rather than being set to black in one pass. Arbitrary oblique planes are displayed using one of two operator selected formats. An oblique plane can be displayed within the volume by selectively "dimming" the voxels around the desired plane or by superimposing a sparse, bright cutting plane over the data. Both types of planes are effected by altering the control fields of the affected voxels by either reducing their intensity or maximizing their intensity. As these operations are performed, the resulting data is composed into a stack within a high-speed buffer. This stack is displayed on the varifocal mirror in back-to-front order synchronously with the vibration of the mirror. The resulting images appear to be a continuous in all three directions, and the operator can look around foreground structures by simply by moving his/her head.

The EDIT process provides interactive modification of the data in memory by thresholding and object tracing. The MANIP process supports addition, subtraction, multiplication, and division of the voxel values in memory by scalar values or other data sets. This process enhances objects as well as interpolates, scales, partitions, and masks out all or portions of the volume. This process also provides the capability for image data set normalization, selective enhancement of objects within the image data, edge contrast enhancement, and “image algebra” functions. Once the regions of interest are isolated using the MANIP, EDIT, MIRAGE and DISPLAY processes, the BIOPSY process can be used to gather values from specific points and areas within the region as specified by the operator. This module displays a set of serial 2D slices extracted from the volume at the desired orientation and provides the capability for point sampling, line sampling, and area sampling with histogram, mean and standard deviation information provided for the area.

The SURFACE process is responsible for identifying and extracting specified surfaces from the volume. The process supports interactive isolation of the surface to be extracted, formation of a binary volume by thresholding the remaining objects according to an operator specified threshold window, and extraction of the binary surface. The extracted surface is stored as a list of voxel faces and is displayed as a set of contours or as a shaded 3D volume on a CRT. The PROJECT process creates user specified projection images of the volume through alteration of the view angle, dissolution range, and dissection parameters[1]. The projection sequences are displayed using the MOVIE process to provide motion parallax, dissolves of tissue, dissection effects, and combinations of these effects to provide additional information concerning the composition of the volume being examined.

4.3.6.1 Shading, Anti-aliasing, and Hidden-Surface Removal Techniques.

Robb's True 3D machine uses the voxel output directly without preprocessing to isolate surfaces within the volume. Shading is not performed in the true 3D imaging process, as the purpose of shading is to convey a sense of 3D relationships and the true 3D display achieves this effect using the varifocal mirror assembly. In the 2D display mode, shading is accomplished by computing the surface normal for each visible face of each voxel. The surface normal, along with the direction of the light and the orientation of the viewer relative to the surface is combined to yield the specular reflectance for the surface. The weighted sum of the specular reflectance term and the distance of the viewer from the voxel face is used as the shading value for the face.

Because of the nature of the 3D display, anti-aliasing is not required. In the shaded surface display mode of operation, anti-aliasing is accomplished by supersampling. In the shaded surface display process, hidden-surface removal is accomplished by using the z-

[1]Dissolution range sets the number of projections performed and the amount of dissolution (fading) to be applied over the sequence. The dissection parameters specify the starting and ending coordinate values of the planes to be sliced away and the number of projections to be performed.

buffer algorithm. Hidden-surface removal would destroy the 3D effect, so is not performed during true 3D processing.

4.3.6.2 Degree of Parallelism and Machine Performance.

There is no parallelism in Robb's True 3D machine. Operations are performed in a strictly sequential manner. For true 3D images on the varifocal mirror, the system can display up to 27 frames every thirtieth of a second (this speed is required to form the true 3D image) at a resolution of 128 x 128 pixels.

4.3.7 Section Summary.

The machines described above represent a wide array of approaches to medical imaging. The architectures range from no parallelism to high parallelism with a correspondingly wide range of performance. The machines also investigate a wide number of approaches, from true 3D to contour extraction with shaded surface display. While these approaches have a wide range of image quality and interactive capability, each has made significant contributions to understanding the processing tasks involved and capabilities required in medical imaging. The MIPG machines led the way in their demonstration of the utility of 3D medical images constructed from extracted contours and surfaces. The algorithms employed in these machines for surface extraction and shading as well as volume representation are used in other machines and demonstrate that image quality need not suffer when the rendering speed is improved. Kaufman and Reynolds demonstrated the usefulness of innovative memory access methods for rapid display of medical images. Farrell demonstrates the usefulness of false color, composited images in a medical imaging environment. Fuchs' two machines establish that the pixel painting bottleneck can be overcome by expending hardware resources at the pixel level which yield significant performance returns. Fuchs and Reynolds demonstrate the validity of a pipeline approach, albeit using different pipelines. Finally, Robb's work provides the basis for modeling the generation of images by a series of cooperating processes by his work in developing the ANALYZE system, and its attendant capability for display of both 2D contour-based and 3D volume-based images. These machines laid the foundation upon which the commercial machines discussed in the next section are laid.

4.4 Commercial Medical Imaging Machines.

This section concludes the presentation of medical imaging machines with a description of the commercial Ardent™ Titan™, AT&T Pixel Machines™ 900 Series™, the Pixar Image Computer™ and Pixar II™, the Stellar® GS series, and the Sun TAAC-1™. As these machines are commercially oriented, their primary aim is the attainment of rapid image rendering speed with acceptable image quality instead of demonstrating new concepts, as is the case for the research-oriented machines discussed in the proceeding section. The descriptions in this section are longer than in the research-oriented machines section owing to the nature of the innovations in the machines. Their innovations range from the areas of rapid memory access and pipelining to the use of proprietary rendering packages with varying capabilities. The architecture of each machine is described in sufficient detail to illuminate its uniqueness, the interested reader is referred to the bibliography for each machine for further information. A final comment before proceeding, these machines are capable of rendering many types of images, not just medical images. As the purpose of this review is a description of medical imaging machines, these other capabilities are mentioned briefly and not assessed for their performance.

4.4.1 The Ardent™ Titan™.

The Ardent™ Titan™ and the Stellar™ GS series, discussed below, are machines classified as personal graphics supercomputers and are not designed specifically to be medical imaging machines. This class of machines combines the computational facilities of a mini-supercomputer, the graphics capability of a high-quality graphics workstation, and the interactive, Unix-based services of an office workstation in one unit. The goal of the Titan architecture is to provide high-quality, interactive graphics dedicated to a single user. Because the graphics processing arena is rapidly changing, the Titan uses a software implementation of a graphics pipeline. This choice was made to provide for rapid implementation of improved image rendering algorithms within the pipeline. However, by opting for flexibility, the Titan encounters a formidable computational burden when performing real-time[1] or photo-realistic[2] volume renderings. To achieve the performance requirements for the system while retaining the implementation flexibility inherent in software, the Titan is designed as a symmetric multiprocessing machine which employs high-speed processors, pipelines, and wide bandwidth between the major components of the machine. The following presentation is drawn from [All88], [Ard88a], [Ard88b], [Ard88c], [Bel88], [Bor88a], [Bor88b], [Bor88c], [Die88], [Eli89], [McL88], [Mir88], [Mol88], [Til88], [Wil88], and [Wor88].

[1]Real-time image display is defined to be a display rate at or above the flicker-fusion rate of the human eye, which is approximately 30 frames per second.

[2]A photo-realistic image is a computer-generated image which has the display characteristics, such as specular reflectance, shadows, and diffuse reflection, which would appear in a photograph.

The processing strategy employed within the Titan is to obtain rapid image rendering through independent parallel operation at the system, component, and thread[1] level, thereby overlapping graphics processing throughout the graphics pipeline. The quest for image rendering speed is assisted by a large vector register file, proprietary pixel and polygon rendering processors, and an optimizing compiler. The use of Unix™ with extensions as the operating system provides support for concurrent operation at the system level and for high-speed I/O processing and disk-striping[2]. At the component level, CPU dedicated pipelines, pipes, arithmetic units, a vector register file, and caches along with interleaved memory[3] enable high-throughput with minimum contention with other system components. At the thread level, synchronization via semaphores and doorbells[4] provides the capability to assign multiple processors to a single computational task.

The supplied compilers support both medium- and fine-grained parallel operation. Medium-grain operation is enabled through automatic determination of the code segments which can run concurrently and allocating these segments to separate threads. Threads are assigned to processors at run-time, permitting a single section of current code to be divided among the available processors in the machine. Fine-grain operation is accomplished by procedure inlining[5] coupled with dependency analysis to perform automatic

---

[1]A thread is a locus of control within a process which consists of a program counter and the thread's register states. The threads of a process share address space and OS context but their registers and stack space are private.

[2]Disk-striping is a file allocation technique which places contiguous blocks of data on separate disk drives, thereby providing simultaneous multiple block reads.

[3]Memory interleaving allows a processor to submit a data request to a memory bank and then proceed on to access the next bank, the processor does not need to wait for the bank to respond, and the banks in the interleave can cycle independently. More interleaves reduces the avarage access time to memory.

[4]Doorbells are used for interprocess signaling via the system bus.

[5]Procedure inlining is a compilation technique which eliminates procedure calls by moving each called procedure to the point in the code at which it is called, thereby placing it in-line for better analysis for vectorization opportunities.

vectorization/strip mining of loops and the detection of integer operations which can be vectorized. The resulting workload can then be distributed among several processors. The compiler also supports the use of compiler directives to direct parallelization of code.

The Titan processing strategy is supported by a hardware architecture which allows simultaneous, independent operation of the CPUs and their dedicated vector/floating point processors. The hardware architecture of the Titan, shown in Figure 4.10, is designed to operate as a 64-bit vector machine comprised of four major components, the Central Processor Subsystem, the Main Memory Subsystem, the Graphics Subsystem, and the I/O Subsystem. These elements are linked together by two synchronous buses, the Read (R) bus and the System (S) bus, each providing 32-bit addressing and 64-bit data paths. The bus does not queue requests, rather the initiating device waits until the servicing device signals its availability before initiating a transaction. The R bus is dedicated to memory vector reading and the S bus is employed for all other reads and writes to memory and for I/O, together they offer an aggregate 256 Mbytes/second data transfer rate for I/O or memory access. The Memory Subsystem consists of from 1 to 4 memory boards and is designed to accept a request from and deliver data to both the R and S bus each clock cycle. Memory interleaving supports up to 16 concurrently active memory transactions, thereby allowing the subsystem to furnish data to the bus at the bus rate of 256 Mbytes/second. If requests arrive at the memory subsystem faster than they can be serviced they are held in a FIFO queue until they can be processed, thereby insuring correct data return order. The Central Processor Subsystem contains between one and four CPUs. Each CPU contains a vector processing unit, a general-purpose integer/control unit, a 32k byte cache, and additional buffers to support rapid CPU operation, up to a peak rate of 16 MIPS. The design of a single CPU is depicted in Figure 4.11. The integer unit has a RISC architecture and performs the CPU control tasks, integer computations, and issues vector/floating point instructions to the vector control unit. The 32k bytes of cache at the integer processor is

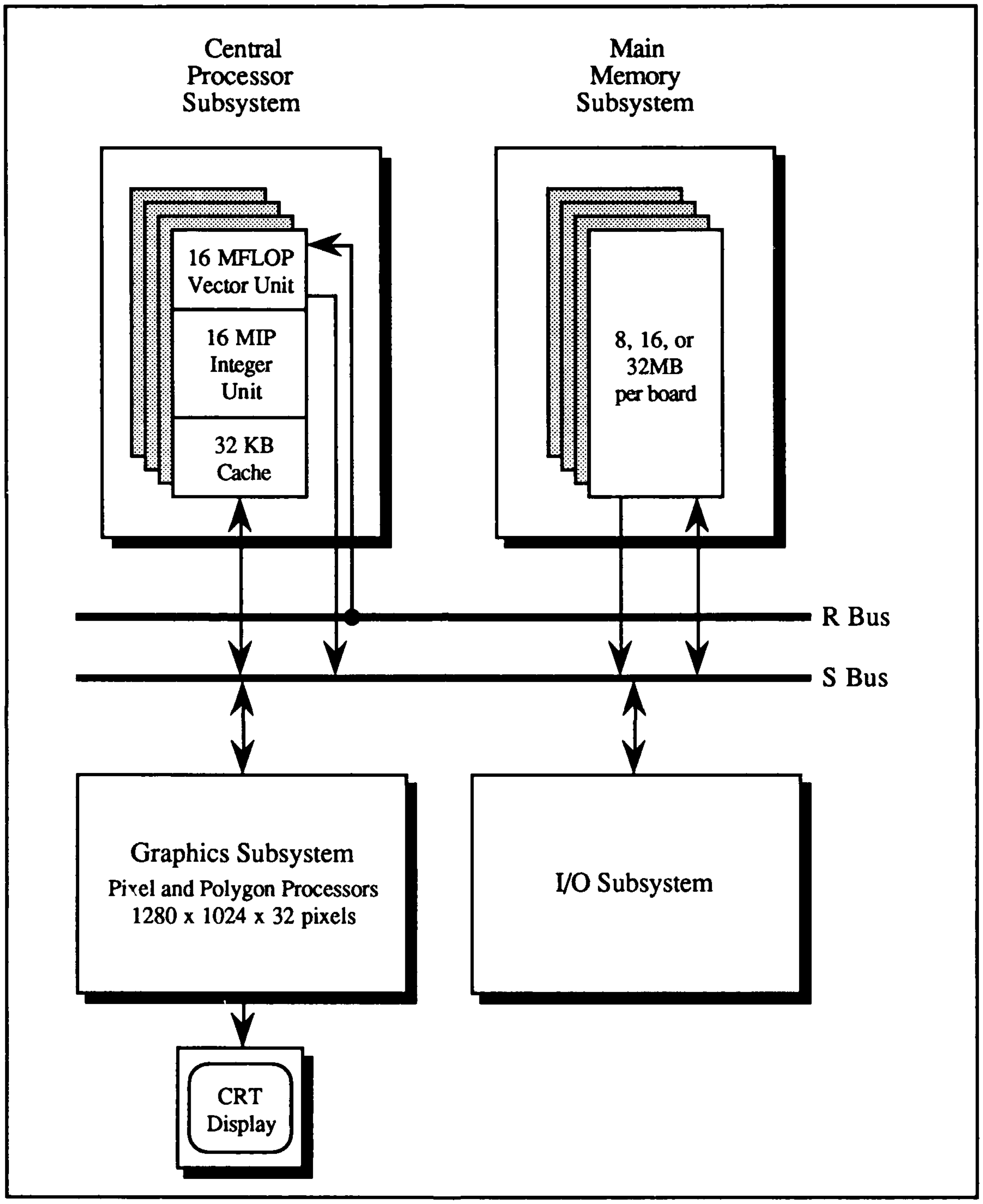


Figure 4.10: Ardent Titan Hardware Architecture (based upon [Die88]).

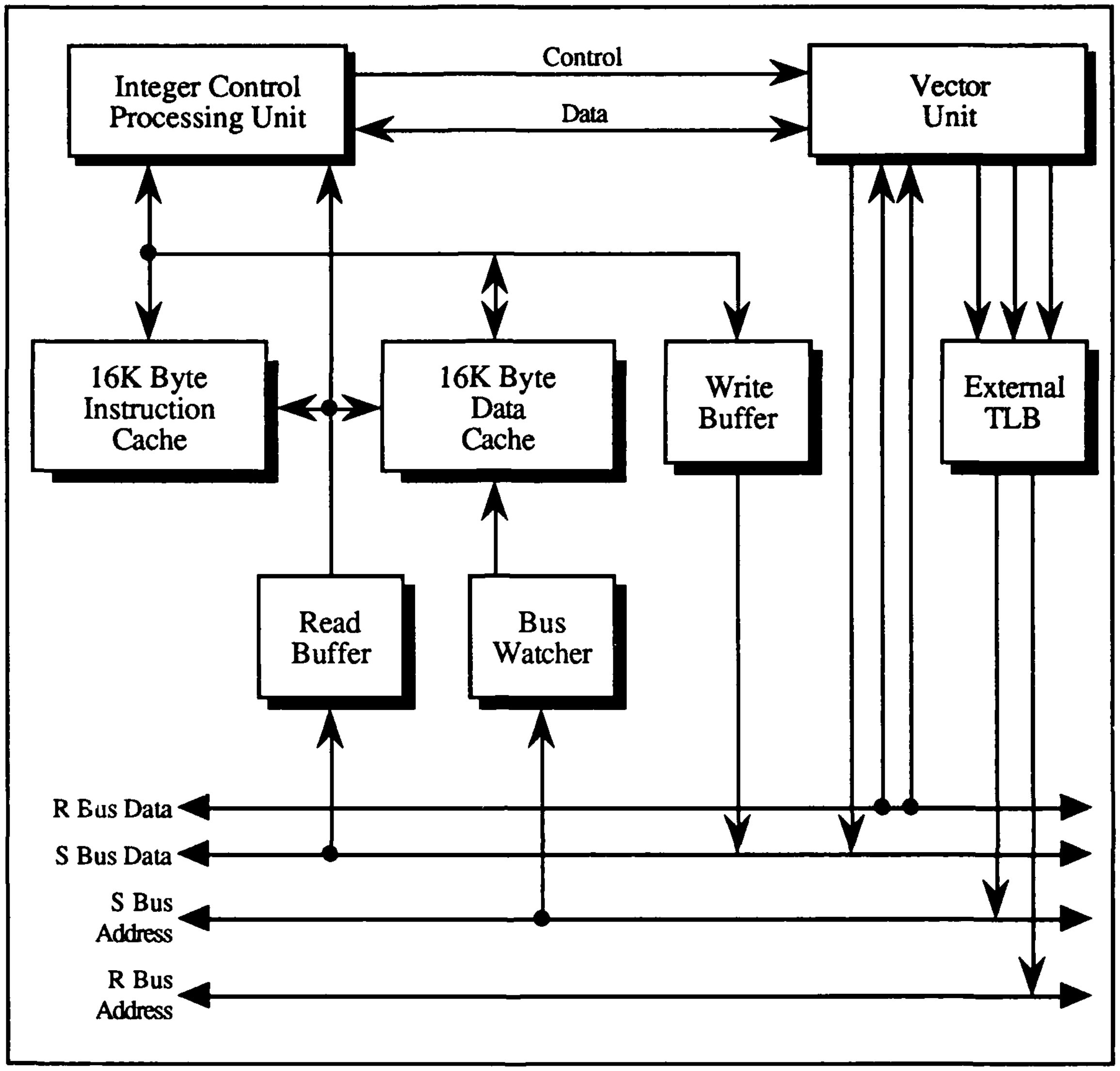


Figure 4.11: Ardent Titan CPU Block Diagram (based upon [Die88]).

equally divided between an instruction and data cache, both of which can be accessed in the same clock cycle. The caches operate independently, using fetch-ahead and a four-deep[1] load pipe to increase the hit rate in both caches and a four-deep deep write buffer and data cache write-through to minimize the chances that the processor must wait during a write. Bus snooping is used to maintain data cache and main memory coherence. When the integer control unit processes a vector or floating-point instruction, the instruction is passed

[1]The Titan reads/writes two words at a time, the four-deep buffers hold eight words.

to the vector control unit via an instruction queue and the integer unit continues processing until the vector unit signals completion of its task.

The vector processing unit (VPU) was specifically designed to perform graphics calculations and is used for both graphics and vector operations. The VPU consists of the vector control unit, the vector register file (VRF) , the vector data switch, the vector data path, and three memory pipes, two for memory loads and one for memory store. The unit is designed to support pipelined execution of vector/floating-point operands, when the components are operating concurrently the vector control unit achieves a peak rate of 16 MFLOPS. A block diagram of the VPU is presented in Figure 4.12.

The vector control unit is responsible for decoding the instructions received from the integer unit, signaling it when an instruction completes, setting up the memory data streams, configuring the data switch and data path, and performing pipeline concurrency control. The vector control unit operates concurrently with the integer control unit, with synchronization performed only when directed by the application being run. The VRF supports registers of variable length, allowing an application to trade off vector size for register file size within a range of 32 registers of 256 8-byte numbers to 8192 registers of one byte numbers. The size of the file is an important factor in the speed of an application, as all operations in the vector unit, except for integer unit synchronization, use the vector-register file as the source and sink for data. The VRF is organized into four banks, with each bank assigned to a thread to increase the probability that required data remains in the file after a context switch, as well as increasing context switch execution speed. The vector data switch is a crossbar switch which serves to route data between the two memory-load pipes, the memory store pipe, the vector register file, and the vector data path. The vector data path can simultaneously perform addition, multiplication, and division on floating point or scalar vectors.

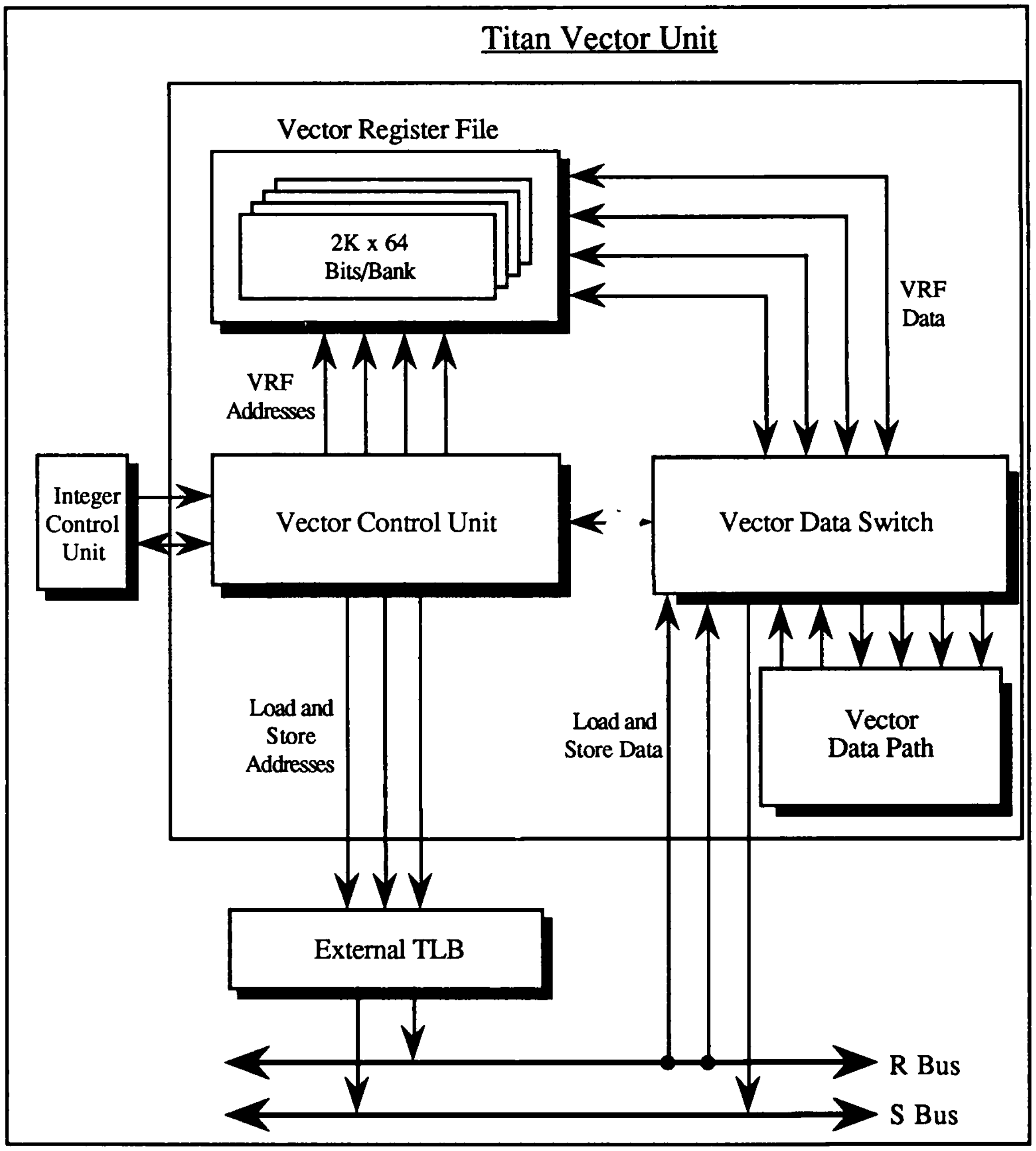


Figure 4.12: Titan Vector Unit (based upon [Die88]).

The graphics subsystem contains the only special-purpose hardware in the Titan system. This subsystem can perform rasterizing scan-conversion using pixel and polygon processors, a 16-bit z-buffer, and shading on a 1280 x 1024 display. The system supports up to 48 image planes in the frame buffer. Pixel processing is accomplished by eight to twelve parallel pixel processors operating upon four scan-lines at a time, using separate processors for each color channel and one additional processor for the z-buffer. Pixel processing is based upon scaling, rotation, and lighting for the image.

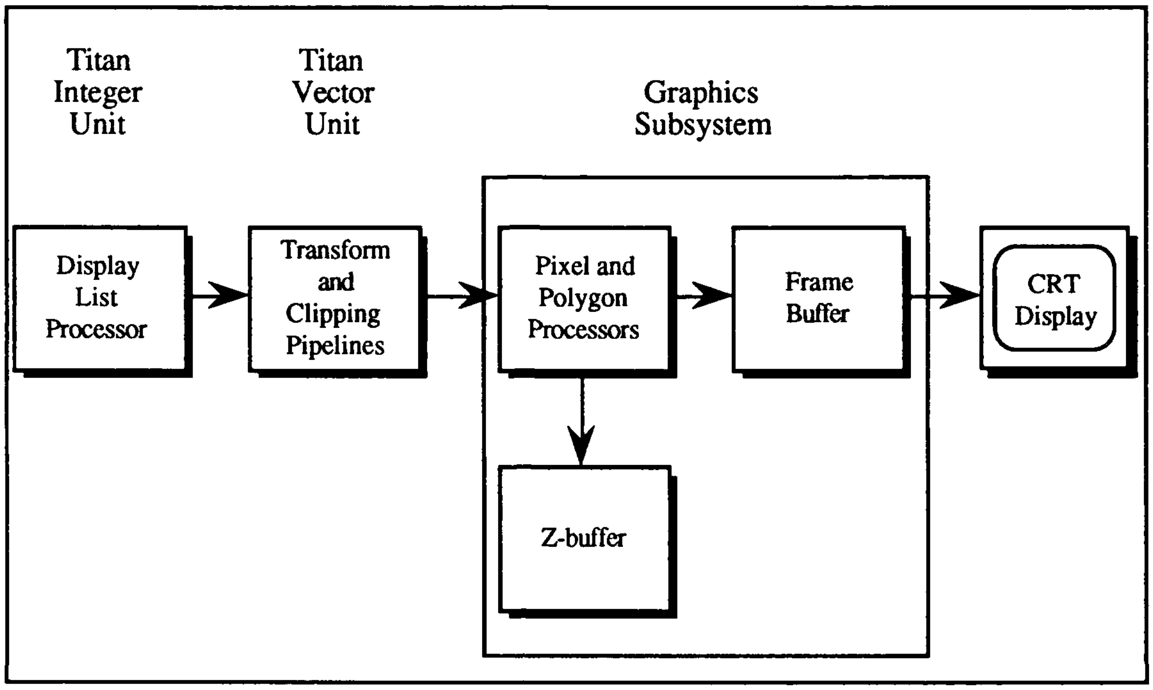


Figure 4.13: Titan Graphics Pipeline (based upon [Die88]).

The graphics pipeline, depicted in Figure 4.13, uses the integer and vector units to drive the graphics subsystem. The integer unit drives the pipeline by processing the display list. When an element is removed from the display list, the element is dispatched to the vector unit for geometric processing, shadowing, reflection and/or ray tracing floating-point calculations. The graphics subsystem uses the resulting values to perform its rendering tasks.

Dore´, the Dynamic Object Rendering Environment, is the object-oriented[1], user extensible software library used to implement image rendering operations above the pixel level. The package supports interactive rotation, pan, and zoom image modifications along with shadows, reflection, anti-aliasing, colored lighting effects, ray tracing, composition,

[1]In Dore´, each geometrical primitive, attribute, device frame, view, etc. is a separate object consisting of a single data structure. Dore´ is not an object-oriented programming environment; objects are used to simplify the management of graphical data.

and transparency. Ambient, diffuse, and specular lighting models are supported for scene shading. When the Titan is operating as a volume rendering pipeline, Dore´ uses the integer and vector units to perform the required transformation, clipping, shading, and pixel writing operations. The Dore´ toolkit stores data in a hierarchy consisting of image primitives, appearance attributes, geometrical attributes, studio objects, and organizational objects. Primitive objects range from simple points, vectors, surfaces and to complex objects such as B-splines and NURBS[1] patches. The description of the object is constructed so that an application can switch between display types, say from wire-frame to surface, without maintaining two display lists or re-defining the scene. Appearance attributes are used to define color, specularity, transparency, texture maps, and environmental reflection for an object. Geometric attributes modify the geometry of primitive objects by transformation, rotation, and scaling. Studio objects are used to define lighting and camera effects for a scene. Organizational objects are used to establish a hierarchical organization for the objects within the scene.

4.4.1.1 Shading, Anti-Aliasing and Hidden-Surface Removal Techniques.

Other than for z-buffering at the pixel level to perform hidden-surface removal, the Titan performs its graphics operations in software, allowing the user to modify the performance and algorithms used in the graphics pipeline to operate upon voxel data in medical images. In this section, the discussion is limited to the functions provided in the standard Dore´ graphics library, with the understanding that these functions can be expanded upon. Dore´ provides support for Phong, Gouraud, and flat shading on either voxel or 2D surface data types. Hidden-surface removal can be accomplished in software using a floating-point z-buffer instead of the integer-based hardware z-buffer. Anti-aliasing

[1]NURBS - *non-uniform rational B-spline surfaces.*

is performed using stochastic sampling when using ray-tracing techniques to render the image or nearest-neighbor sampling when performing standard volume rendering.

4.4.1.2 Degree of Parallelism and Machine Performance.

The Ardent Titan is a highly parallel MIMD machine, with the potential for 16 concurrent memory transactions, an add, multiply and divide operation per vector unit, and one integer operation per integer control unit each 62 nanosecond clock cycle. The performance of the machine is further enhanced through the use of pipes and buffers. The peak system performance in the Titan of 200,000 Gouraud-shaded polygons/second is directly attributable to the high degree of parallelism within the architecture. A 256 x 256 x 70 voxel scene can be rendered at a near real time rate of 15 frames/second.

4.4.2 AT&T™ Pixel Machine 900™.

The AT&T Pixel Machine 900 series is a general-purpose machine designed to provide high performance through the use of both coarse-grain pipelining and fine-grain computing arrays. The pipeline is used to perform serial image rendering tasks such as object generation and geometric transformation. The processor array computes individual pixel values based on the output of the pipeline and provides high-bandwidth access to a distributed frame buffer. All image processing algorithms, such as image transformation, shading, and hidden-surface removal are software encoded. Like the two Pixar machines discussed later, the Pixel Machine requires an attached host machine for support. In this case, the host[1] is responsible for networking tasks, the user interface, and the program development environment. In addition, the host supports the Pixel Machine by translating the high-level function calls into the Pixel Machine, loading the pipeline and array with

[1]The host can be either a Sun™ 3, Sun™ 4 or Silicon Graphics 4G.

execution code, and for managing iterative functions. The machine architecture and performance description in this section is based upon [Att88a], [Att88b], [Att88c], [Att88d], [Att88e], [Att88f], and [Att89].

The design objectives of the Pixel machine call for the use of 32-bit floating-point computations for computational accuracy and a 32-bit voxel representation with large image memories for medical image generation. Image rendering computations are performed on floating-point processors each executing code at the rate of 5 MIPS and 10 MFLOPS. Image generation algorithms are software encoded to permit user modification of these functions. The overall machine architecture is presented in Figure 4.14. The major components of the Pixel Machine

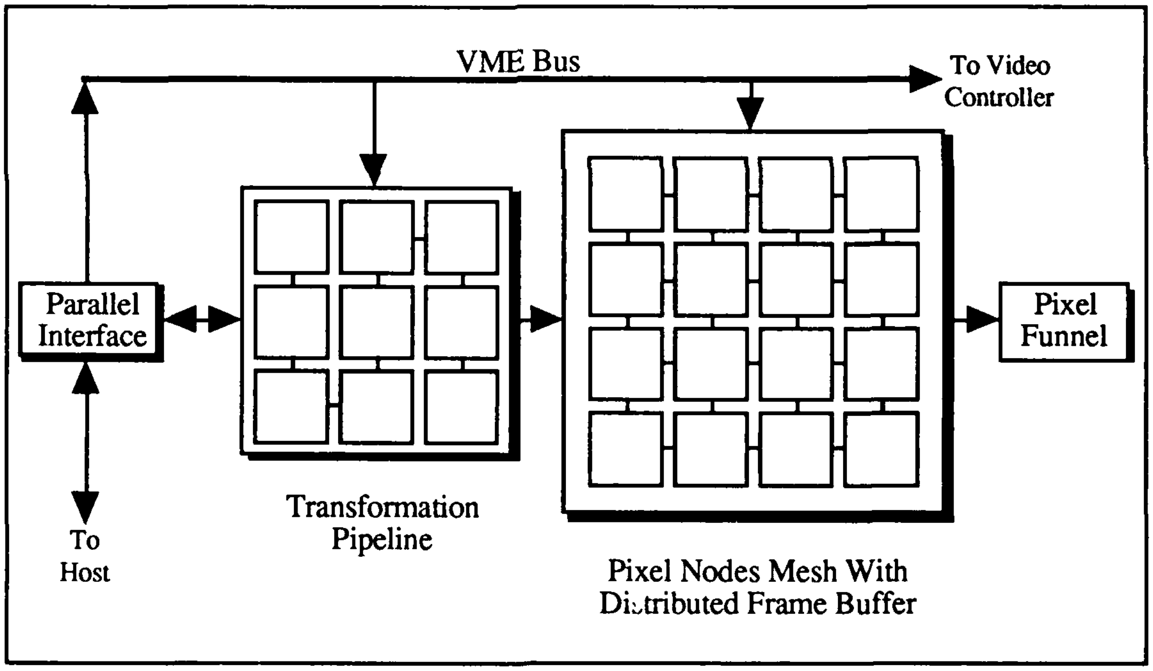


Figure 4.14: Pixel Machine Architecture (from [Att89]).

900 series machines are the host, the Transformation Pipeline, the Pixel Nodes, the Pixel Funnel, and the VMEbus™. The host computer system accesses the Pixel Machine via the VMEbus as a 64k byte block of memory, with this memory mapped into user process space. The parallel I/O interface for the pipe and array nodes, the FIFO queues on the

pipeline, the colormaps, and video control registers are mapped directly into this 64k memory block.

The primary recipient of host communications is the Transformation Pipeline. The Transformation Pipeline can be configured as a pipeline of either nine or eighteen nodes, with the 18 node configuration capable of operation as one long pipe or two shorter parallel pipes. Because the Transformation Pipeline is designed to perform serial image rendering operations, rotation, translation, scaling, and projection operations are most efficiently performed there. The pipeline nodes use the WE™ DSP32 digital signal processing chips as their CPU. The configuration of each node is depicted in Figure 4.15. The static RAM at the node is used for instruction

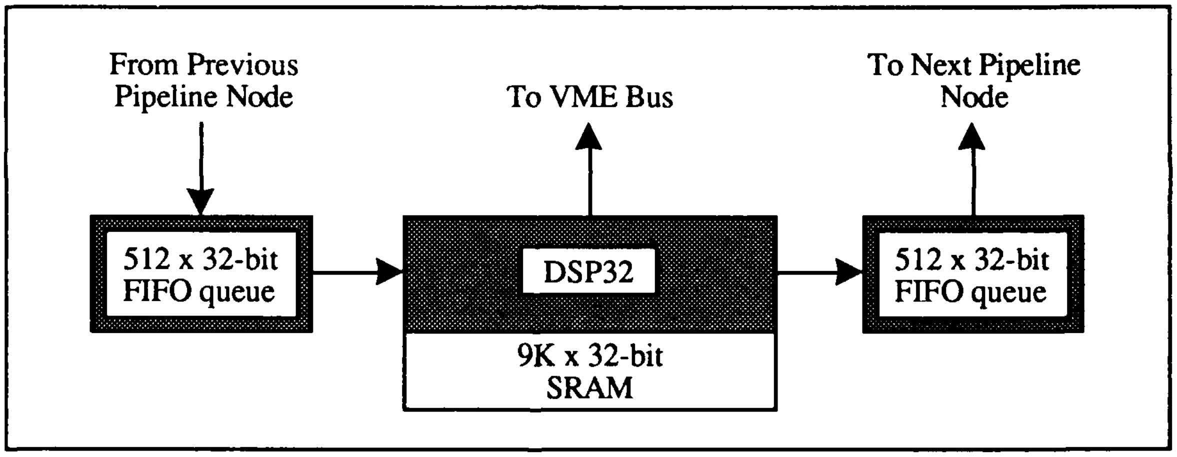


Figure 4.15: Pixel Machine Pipeline Node Configuration (based upon [Att89]).

and data memory and the queues are employed for communication with neighboring nodes in the pipeline. The input to the first node in the pipeline is provided across the VMEbus by the host. The first node then performs the one or more serial operations assigned to it by the user and passes the result to the next node in the pipeline. The image is staged down the pipeline as subsequent pipeline nodes remove data from their input queue, with each stage performing an independent set of operations on the data. The last pipeline node broadcasts its output to all of the Pixel Nodes and optionally back to the host via the VMEbus.

The Pixel Nodes are arranged as a mesh of processors all of which are connected to a distributed frame buffer which supports screen resolutions of either 1280 x 1024 or 1024 x 1024 pixels. The configuration of a single node is portrayed in Figure 4.16. The mesh can be

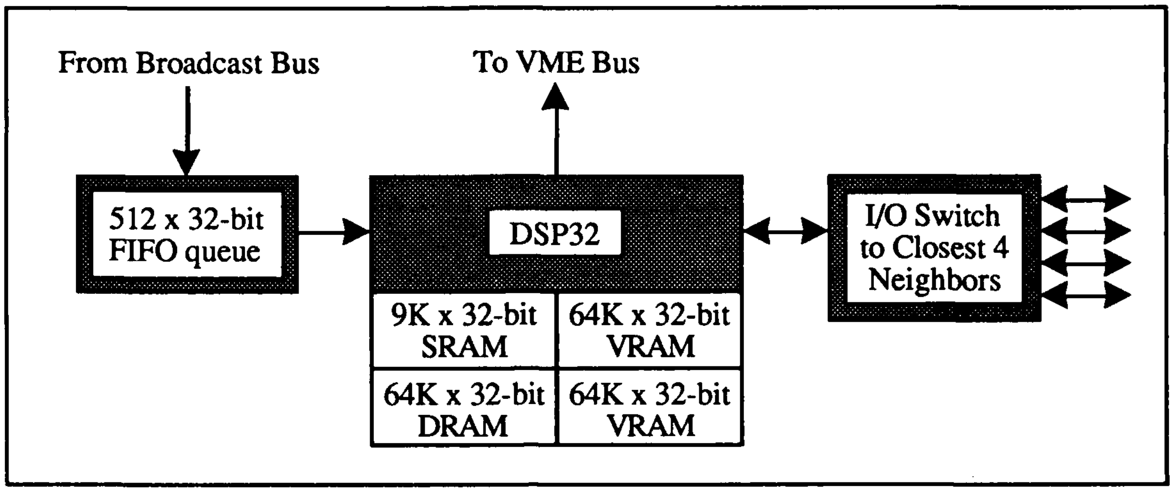


Figure 4.16: Pixel Machine Pixel Node Architecture (based upon [Att89]).

populated with 16, 20, 32, 40, or 64 Pixel Node processors, with each processor capable of serial communication with its nearest neighbors. The CPUs for the processors in the mesh are the same WE™ DSP32 digital signal processing chips used in the Transformation Pipeline. The Pixel Nodes are configured differently from the transformation pipeline nodes. Pixel Nodes have as much static RAM for instruction storage as pipeline nodes, and have an additional two banks of 64k x 32-bits of video storage, thereby providing each node with enough memory to hold its share of two full screens of 32-bit frame buffers. The largest mesh configuration provides 32 Mbytes of frame buffer memory and 16 Mbytes of data memory . The two banks of video RAM and one bank of dynamic RAM can be used to hold floating-point z-buffer values, code segments, or pixels in a floating-point representation. The video RAM is primarily used to store each of the 16-bit[1] signed

[1]In the currently available Pixel Machine models, 8 of the 16 bits are populated with memory, the remaining bits are reserved for future expansion.

red, green, blue, and alpha channel components of each pixel in the portion of the distributed frame buffer at each Pixel Node. To accommodate the various processor mesh configurations without software alteration, each Pixel Machine uses either 64 or 80 virtual Pixel Node processors mapped to the actual mesh, with one or more virtual processors assigned to a given Pixel Node.

The virtual processors operate upon 256 x 256 x 32-bit subsets of the video and dynamic RAM at each processor. These subsets, called sub-screens, are permanently assigned to a virtual processor, thereby permitting each virtual processor to access the data for its subscreen without conflict. Image display is performed by computing the pixel color and intensity, shading the surface(s) represented in the pixel, anti-aliasing, and image compositing. To accomplish the image display tasks, the virtual processors perform point, line, and polygon rasterization, ray-tracing, z-buffer calculations on polygons and spheres, adaptive histogram equalization [Piz87], orthographic and perspective projections of the data, and the output of individual pixels to the Pixel Funnel.

Each of the Pixel Nodes writes to an assigned portion of the frame buffer, with the screen pixels interleaved among the processors in the mesh as shown in Figure 4.17. To distinguish between pixels at a processor and pixels on the screen, two spaces are defined: processor space and screen space. The coordinate system defined within the sub-screens is called processor space, the coordinate system on the display screen is called screen space. Because the virtual processors do not operate on contiguous portions of the screen, processor space cannot be directly mapped to screen space. The mapping relation defined on the pixels in processor space assigns screen pixels to each of the processors in the mesh based upon the location of the processor within the mesh[1]. This mapping does not pose a

[1]If N_x and N_y indicate the number of processors per row and column of the mesh, (i,j) represent the coordinates of a given processor space pixel, and P_x and P_y represent the row and column indices of a particular processor in the mesh, then the screen pixels at coordinates (x,y) assigned to the processor (P_x, P_y) are computed by: $x = iN_x + P_x$; $y = jN_y + P_y$.

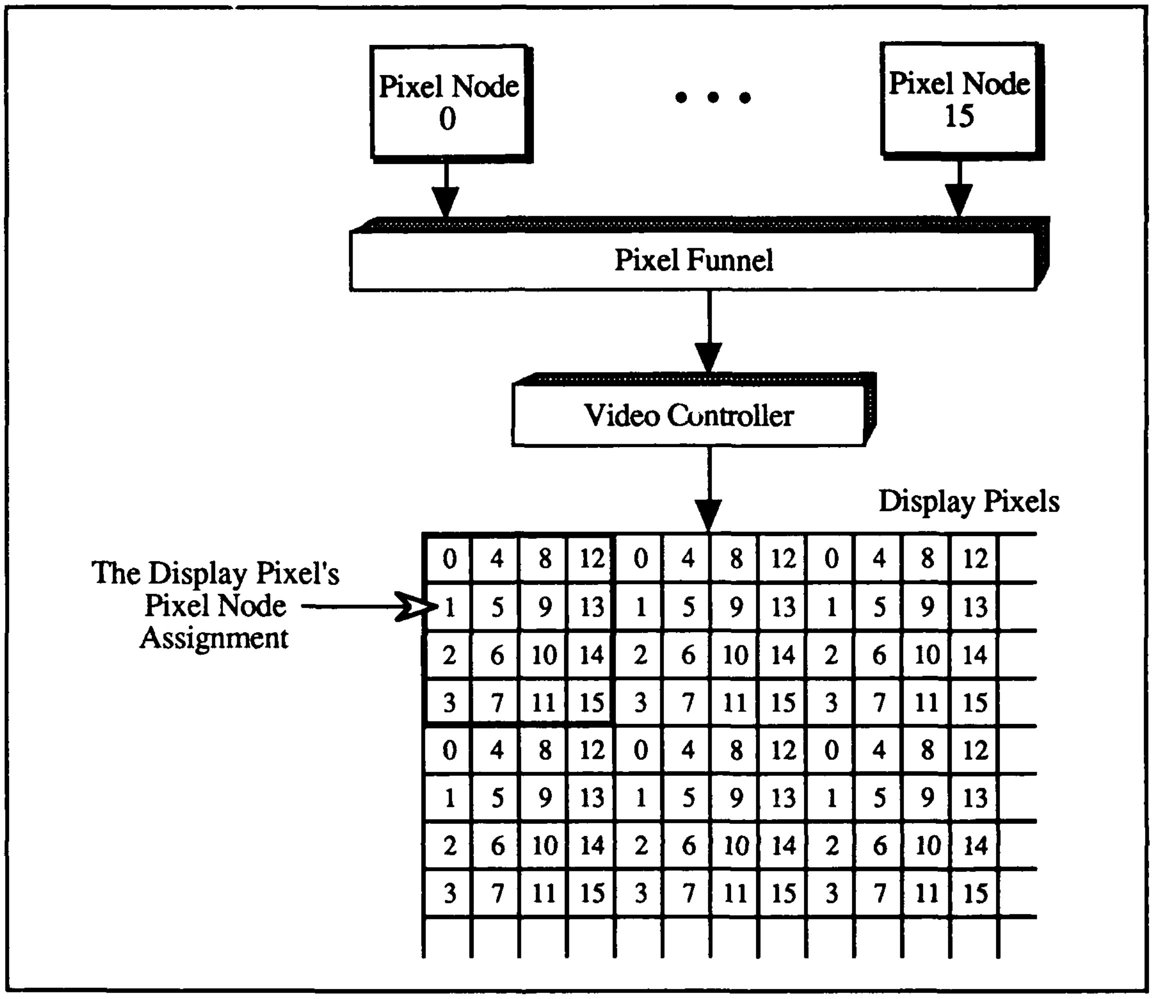


Figure 4.17: Pixel Node to Display Pixel Mapping (based upon [Att89]).

problem for algorithms, such as ray tracing and fractal generation, which do not assume some form of pixel-to-pixel coherence. Algorithms which have a pixel-level contextual basis and require contiguous pixels to be in the same processor, as is the case for gradient shading and normal-based contextual shading, are implemented by performing additional communication within the mesh. After each processor has computed the pixel values for the contiguous pixels assigned to it, the output pixels are routed to the processor in the mesh normally responsible for the interleaved display of the pixel. The output from each of the Pixel Nodes is sent to the Pixel Funnel, which accomplishes the processor space to screen space mapping and sends the rasterized output to the video controller for display.

4.4.2.1 Shading, Anti-Aliasing and Hidden-Surface Removal Techniques.

As the Pixel Machine does not encode any image processing algorithms in hardware, the algorithms employed for rendering medical images are left to the user's discretion. The capabilities discussed here are limited to the image rendering code supplied with the machine. Two libraries, RAYlib™ and PIClib™, are provided which perform medical image rendering tasks such as shading, anti-aliasing, and hidden-surface removal. PIClib provides flat, Gouraud, and Phong shading software, texture mapping, and up to 150 colored light sources for voxel shading. Hidden-surface removal can be performed using either z-buffer or backface surface-removal algorithms. Anti-aliasing is accomplished by supersampling, with the number of samples and the filter shape defined by the user. Supersampling is accomplished by rendering the image multiple times. At the conclusion of each rendering, the red, green, and blue contents of the frame buffer are multiplied by the filter coefficient that corresponds to the pass just completed. These products are accumulated, and after the final pass the results are converted from floating-point to the 32-bit integer pixel format and displayed.

RAYlib provides the common ray tracing functions such as shadows, reflection, texture mapping, and transparency. Multiple arbitrary-colored light sources are supported for combinations of point, direct, and ambient lighting. Anti-aliasing is accomplished using adaptive stochastic anti-aliasing. Given the similarity between the Pixar data format (described below) and the Pixel Machine data format the volume rendering and volume classification techniques developed for Pixar should be implementable as Pixel Machine routines.

4.4.2.2 Degree of Parallelism and Machine Performance.

Pixel Machines is highly parallel, with the capability for each node of the pipeline to compute a different portion of the image rendering transformation while the Pixel Nodes mesh displays the most current output from the pipeline. The high-end model has a peak performance of 410 MIPS and 820 MFLOPS using 18 pipeline nodes, 64 mesh nodes and a 32 Mbyte distributed frame buffer. A 256 x 256 x 60 slice image can be rendered from voxels and colored using a single ray to shade each pixel in 10 seconds.

4.4.3 PIXAR's™ Pixar Image Computer™ and Pixar II™.

The Pixar Image Computer and the Pixar II are commercially available general-purpose graphics computers which function at the personal graphics supercomputer level. Unlike the Ardent™ Titan™, and the Stellar™ GS series, the Pixar machines require a host computer for non-image computing functions such as network access, a program development environment for the Chap (*Cha*nnel *P*rocessor) C compiler and Chap assembler, and for a user interface. The following material is drawn from [Car84], [Cat84], [Coo84a], [Coo84b], [Coo87], [Dre88], [Duf85], [Lev84], [Pix88a], [Pix88b], [Pix88c], [Pix88d], [Pix88e], [Por84], and [Spr86]. The description of these machines is limited to the image processing algorithms supplied with the Pixar machines. However, since the algorithms are software encoded, the user is able to modify them relatively easily.

The Pixar Image Computer and Pixar II software and hardware designs were driven by the requirements found in image composition in film special effects and animation. The techniques employed in both Pixar machines for rendering, anti-aliasing, image compositing, rotation, scaling and shading are described in [Cat84], [Car84], [Coo84a], [Coo84b], [Duf85], and [Coo87]. While not all these techniques are employed in medical image formation, the concepts developed to perform them are the basis to the approach employed by Pixar for medical image rendering. A complete discussion of these techniques is beyond the scope of this paper. The operation of the machines is image[1]

[1]Images are defined to be a collection of picture data organized into a rectangular array.

oriented in the sense that operations are applied to images, scan lines, windows, and individual pixels. The influence of the image compositing process is strongly reflected in the pixel data format. The pixel data structure consists of the three 12-bit color channels (red, green, and blue) and 12-bit transparency channel (alpha) required for forming an image. The alpha channel is used to provide a weighting value for the other three channels which is used to combine their color information with color information in the other elements in the image. This four-channel format is also suited to the display of voxel data, with the red, green, and blue channels serving to hold three-dimensional coordinate values and the alpha channel holding the density/intensity value for the voxel. This data structure was chosen because it permits simultaneous operation upon the four components of each pixel by the four component parallel processing unit, the Chap, that is the basic processing unit of the Pixar computer systems.

The design objectives of the two Pixar machines are to maintain a general-purpose hardware implementation, to exploit SIMD parallel processing at the pixel level to obtain high-speed image generation, to provide hardware support for, but not implementation of, graphics algorithms, and to allow for expandability through the use of modularity in the design of the machine. These objectives are reflected in the design of the machines presented in Figures 4.18 and 4.19.

The basic Pixar Image Computer has one Chap running at 85ns, 1 video board, one memory controller, and three 8Mbyte (24Mbyte total) memory boards, with the option of equipping the system with three 32Mbyte memory boards for a total of 96Mbytes of memory . The system is expandable to three Chaps and six memory boards for a total of either 48, 120, or 192 Mbytes of memory.

The basic Pixar II consists of one Chap and one Frame Store Processor[1] with 12 Mbytes of image memory. The Pixar II can be expanded to two Chaps and three memory

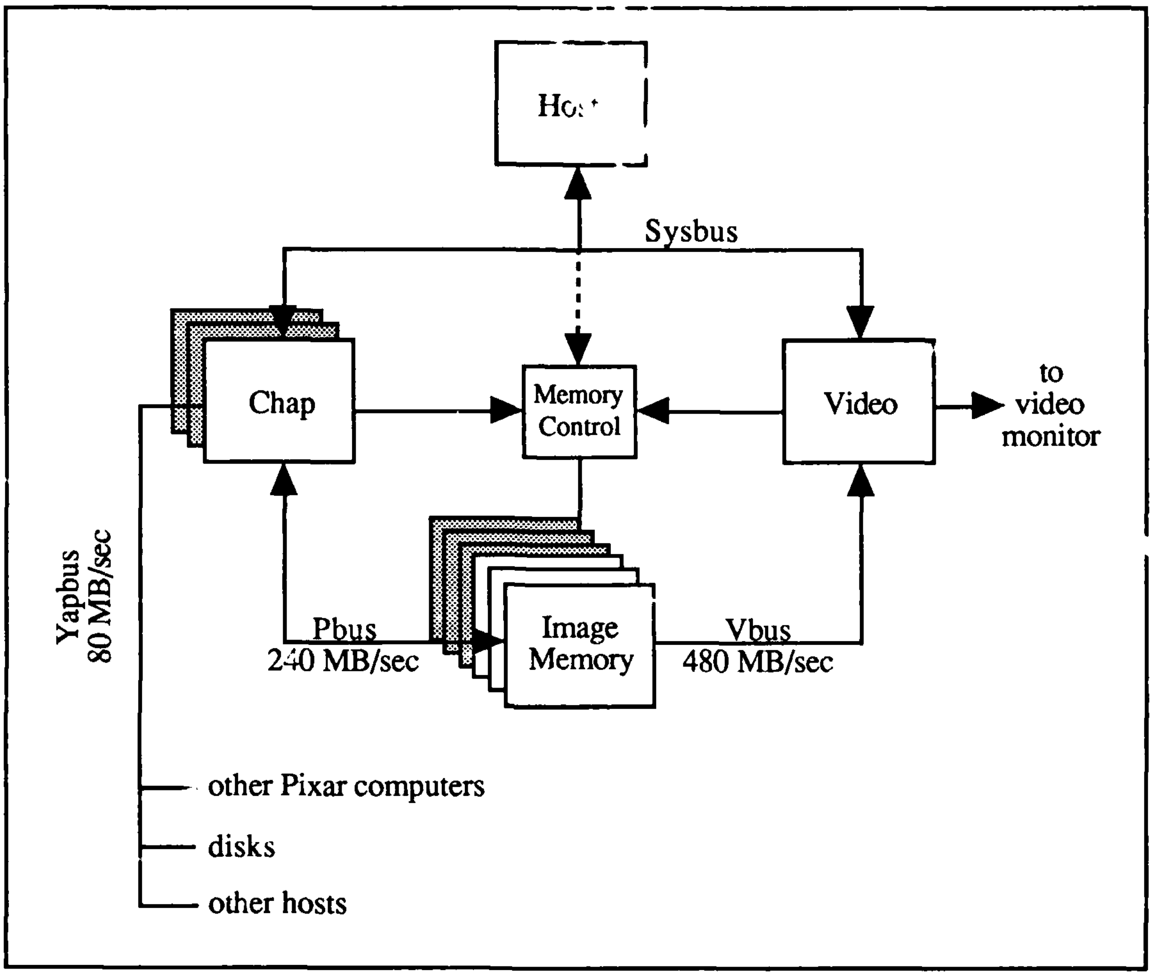


Figure 4.18: Pixar Image Computer Block Diagram (from [Pix88c]).

boards, which can be a combination of Frame Store Processor (FSP) memory, Frame Store memory (FSM), and Off-screen Memory (OSM)[2]. Both the Pixar Image Computer

[1]The FSP board contains a video display processor that generates the analog raster image and memory control circuitry which allows Chap and host memory access parallel with video display operations. The memory appears as a linear array of 32 x 32 four-component pixels called tiles.

[2]The FSM on the FSP contains 12 Mbytes of memory and the OSM board contains 48 Mbytes of memory. Neither the FSM nor the OSM contain display hardware. The OSM's memory model is identical to the FSM's, the difference in the boards being that an image stored in an OSM must be moved to a FSP board for display whereas an image stored in an FSM is moved within the same board to a FSP for display.

and Pixar II systems can support 1280 x 1024 resolution, 48-bit color frames at up to 60 frames/second. The Pixar II also supports monochrome images at 2560 x 2048 pixel resolution.

The host computer[1] is connected through its I/O bus to the Pixar Sysbus (System

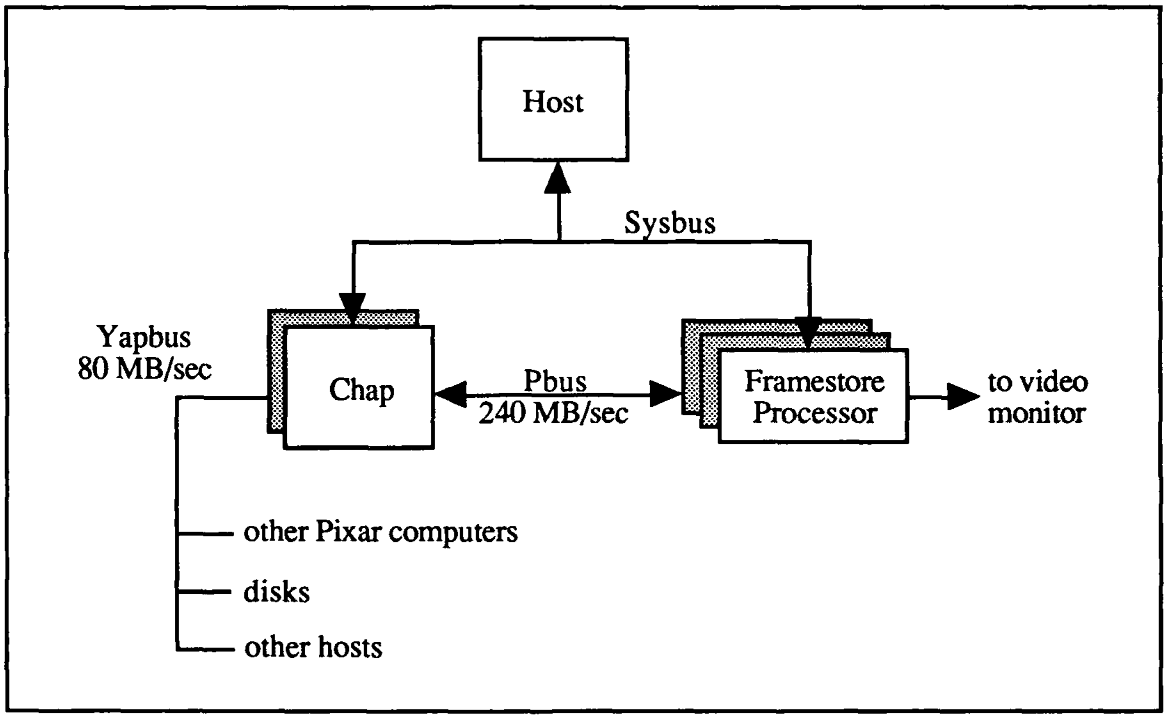


Figure 4.19: Pixar II Block Diagram (from [Pix88c]).

(bus) using a 16 bit wide, 2M byte/second channel for transmitting address and data packets between the host and the Pixar machine. The interface between the host and Chap is supported with 16 words of shared memory for parameter passing and 4096 virtual register locations in the Sysbus, with the added capability for allowing the Chap to interrupt the host as well as vice-versa. The Sysbus is used for host access to the control unit, the address generator, runflags, and memory.

The four Chap ALU processors are tightly coupled to the Scratchpad image memory. Each Chap communicates with peripherals or other Chaps using 64 bit transfers

[1]The host can be a Sun™ 3/110, /140, /150, /160, /180, /260, /280, Sun™ 4, Silicon Graphics™ IRIS™ 3100, Silicon-Graphics 4D computers, or Digital Equipment Corp. MicroVAX™ II/Ultrix™ or VMS.

over the 80 Mbyte/second Yapbus (Yet Another Pixar Bus). Chap communication with memory is supported by the Pbus (Processor Access Bus) at 240 Mbytes/second. The Pbus is a 4 pixel (192 bit) wide bus which moves bursts of 16 or 32 pixels between Chap and picture memory. The Pbus automatically extends 12-bit floating point data to a 16-bit floating point representation to permit an exact representation of 0 and 1. The Chap communicates with peripherals and other computers via the Yapbus, receives instructions from the host, and performs the image processing operations. The memory controller handles scheduling of data transfers to/from video memory from/to the Chaps.

The Scratchpad memory is used for program data storage of up to 16K pixels (16 1K scan lines) for use by the Chap processor. There are four segments within the Scratchpad memory, with each segment used to store one channel of the pixel word. The Scratchpad is connected to the Chap with a crossbar switch which supports four different memory access modes: pixel, broadcast, index, and component. Pixel mode provides simultaneous read/write access to all four channels of the pixel. Broadcast mode allows the four Chap ALUs to receive the same channel memory element from one of the four Scratchpad memories. Indexed mode allows each ALU to access a different address in the Scratchpad. Component mode reads/writes a single channel from four consecutive pixels, and is used for applications which do not operate on all components of the same pixel in parallel. The heart of the system is the Chap, a 4 ALU, one multiplier machine controlled by a Instruction Control Unit (ICU), which functions as a 4 operand vector pipeline operating at a peak rate of 40 MIPS (10 MIPS/ALU). A simplified model of the data paths available between the Chap components and the remainder of the Pixar machine is diagrammed in Figure 4.20.

The four ALU elements of the Chap and the multiplier are connected to the Scratchpad through a crossbar switch. The switch allows the multiplier and ALU to operate in parallel, with the four Mbuses used to load data into the multiplier and the four Abuses used to load data into the ALU. The Sbus (Scalar bus) is used by the ALU and

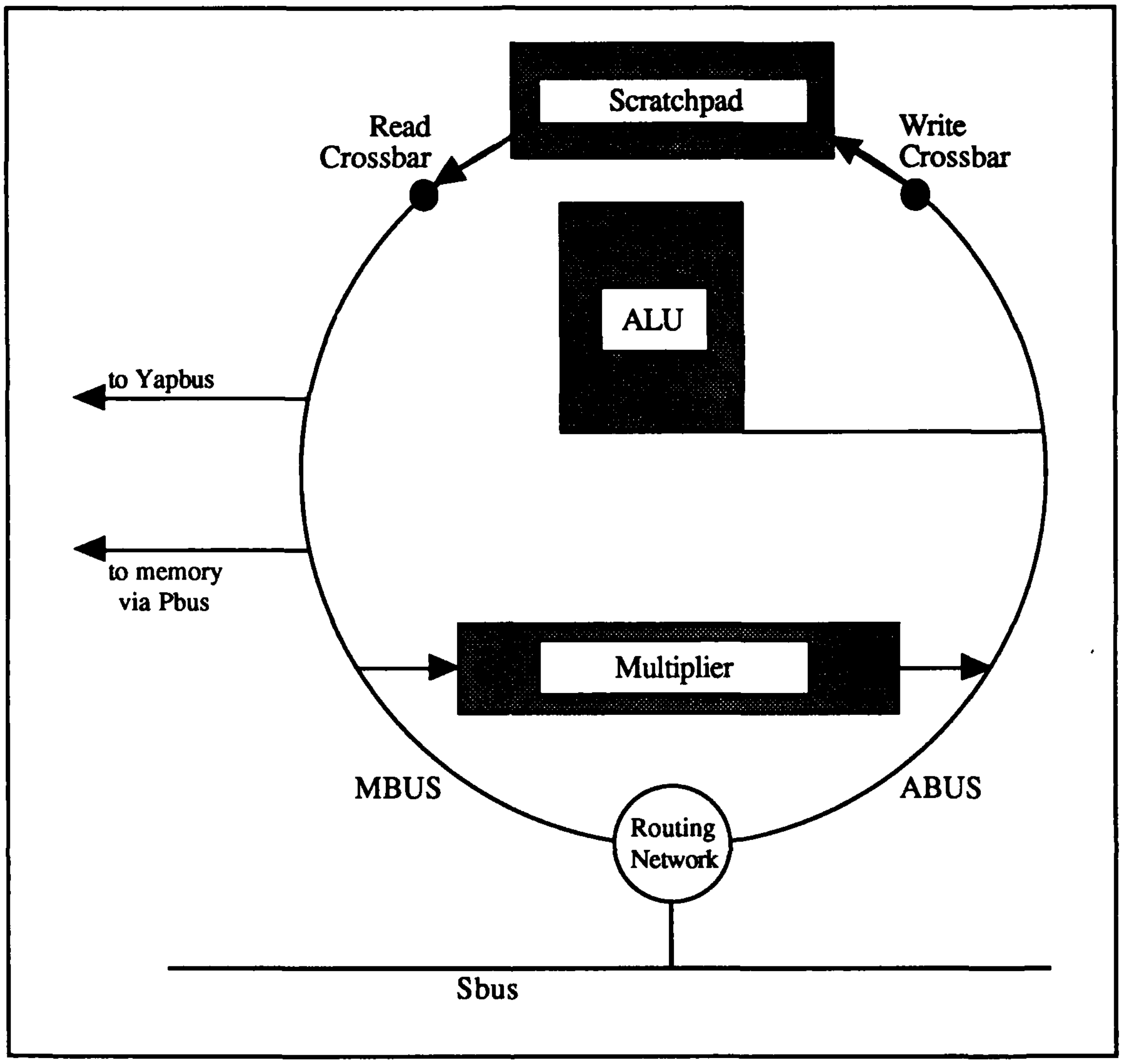


Figure 4.20: Chap Programmer's Model (from [Pix88c]).

multiplier for reading and writing single pixel channels and by the ICU for distributing instructions to the Chap components. Each Chap is capable of processing two major data types: 12-bit, 4-channel integer (the pixel format) and 12-bit single channel integer; 32-bit long integers and 32-bit IEEE floating-point numbers are processed by Chap software routines. The ALUs are dedicated to performing 16-bit integer arithmetic. During program execution, the Chap can switch between performing operations on one channel of each pixel and performing the same operation on the same channel of each pixel. The ICU is used to control the ALUs, multiplier, buses, and the ICU itself as well as calculate Scratchpad memory addresses. The address space for ICU program memory consists of

64K words of 96 bits each and is completely separate from Scratchpad memory. The ICU employs a runflag register and an activity stack to control individual ALU activity. The runflag register maintains the current status of all the ALUs, and is set based upon the instruction to be executed and the current status of the four ALUs (disabled/enabled). The activity stack is employed to maintain runflag register contents at different program levels.

For medical imaging applications, the data set is comprised of voxels organized into slices. Each voxel can be classified by its coordinates, color, opacity, and/or refractive index during the course of processing the data volume. Processing begins by classifying the voxel-based medical image data into material percentage volumes. Each material percentage volume is composed of voxels. Each voxel in a material percentage volume is represented by a pixel, whose alpha channel value is determined by the amount of a given material (such as bone, fat, or air) present in the original image voxel. The voxels can be classified based on some given input data or by using a probalistic classification scheme. The composite volume for a given set of appearance conditions is recovered by forming other voxel-based representations of the original space and then composing all of the representations into a final image. The composition is achieved by combining the material percentage volumes' alpha channel values with the alpha channel values stored within voxel-based representations for the matte volume, color volume, the opacity volume, the density volume, and the shaded color volume. By combining the alpha channel values for the voxels in each volume type which were assigned based upon color, shading, matte, opacity, and density effects desired in the final image, a single composite view of the original image space is rendered. Each type of volume fulfills a specific purpose in the volume rendering system. Matte volumes provide the capability for passing an arbitrary cutting plane or shape through the volume or for changing material properties in a region so as to highlight specific material properties. The color and opacity volumes are themselves composite volumes formed using the material percentage volumes and the color and opacity values assigned to each material. The alpha channel value assigned to each voxel in the color and opacity composite volumes is determined in a two-step process. First, the product of the alpha channel for each voxel in a given material percentage volume and the color/opacity assigned to the material is calculated, and then these products are summed to yield a single alpha channel value for the composite voxel.

Boundaries between materials are determined by applying a three-dimensional density gradient to a density volume. The density volume is another composite volume computed from the sum of the product of each of the material percentage volume voxel values and the density assigned to the corresponding material. The gradient is largest where rapid transitions between materials with different densities occur. The gradient vector is calculated between each voxel and its neighbors and is stored in two separate volumes, the surface strength volume which contains the magnitude of the gradient and the surface normal volume which contains its direction. The surface strength volume is used to estimate the amount of surface present. The surface normal volume is employed in shading calculations. The shaded color volume is formed by composing the surface normal volume, the surface strength volume, the color volume, given light position(s) and color(s), and viewer position using a surface reflectance function. The shade assigned to each voxel is based on both surface scattering and emission, where the amount of emitted light is proportional to the amount of luminous material in the volume. The amount of luminous material can be assigned to each voxel based on the percentage of each material type present within it using the material percentage volumes. The formation of a medical image requires that the shaded color volume be formed and then this volume is transformed according to user inputs and resampled for display. Not all of the above mentioned volume types need be formed for every image, the type of data to be viewed and the representation desired determine the volumes formed and hence the total time required to form the final image.

4.4.3.1 Shading, Anti-Aliasing and Hidden-Surface Removal Techniques.

Shading is accomplished using a single light-ray to depict changes in light intensity as light travels from the back to the front of a voxel. The change in light intensity is modeled by the amount of light absorbed by material in the voxel, the luminosity of the material (which is directly related to the amount of light emitted by the voxel), and the voxel surface, which can scatter the outgoing light ray. Surface shading is accomplished by subdividing the voxel into two parts separated by a surface region. The color of a translucent voxel, which is the emitted light, is determined by composing the colors of each of the three regions weighted by the light absorption characteristic and the intensity of the light for each region. The reflected surface color is determined based upon the local surface normal, the strength of the surface, the color of the surface, the direction to the light source, the color of the light source, and the position of the viewer. The surface color is assumed to be the color of the material behind the surface. The region behind the surface is determined by examining the density gradient along the viewing direction. The color of the reflected light is determined by composing the color of the surface with the color of the light after specular and reflective shading functions have been applied to the color components.

Hidden-surface removal is performed by laying down the slices of the shaded color volumes in back-to-front order. Anti-aliasing can be accomplished using either of two different methods. The original volume can be sampled above the Nyquist limit[1] and maintained at this resolution throughout the image processing until the final image is formed or by low-pass filtering[2] of the final image.

---

[1]The Nyquist limit is the highest frequency at which an image can be sampled without aliasing occurring in the reconstructed image.

[2]Low-pass filtering suppresses the high frequency components of the image. Intuitively, this type of filter removes the components of the image which are changing in less than a diameter of a pixel.

4.4.3.2 Degree of Parallelism and Machine Performance.

The Pixar machines perform parallel operations at the pixel level. The amount of parallelism is determined by the number of Chaps present in the machine and the type of processing being performed. Given that there can be no more than three Chaps in the machine, no more than twelve pixel level calculations can be performed simultaneously. The Pixar machines' processing speed comes from its use of algorithms which require simple calculations at the pixel level and from processors which can push pixels through the machine very quickly, leading to high throughput in the number of pixels computed each second. The time required to form a medical image from a 256 x 256 x 256 voxel volume can vary from a few minutes to an hour depending upon the number of slices in the volume and the number of different volumes required to be formed to render the desired final image.

4.4.4 Stellar® GS series.

The Stellar® GS series of graphics computers are designed to operate as a personal graphics supercomputers, with the graphics processing pipeline constructed to provide 3D rendered images of geometric primitives with a capability for 3D volumetric processing. This machine, like the Ardent™ Titan™, seeks to overcome the computational hurdle in volume rendering through high-speed processing along with wide-bandwidth buses and datapath interconnections instead of special-purpose hardware. This approach results in implementation flexibility for 3D imaging, as the user has the choice of using vendor-supplied software in the graphics pipeline or implementing different algorithms in any stage of the pipeline and relying on the vendor's software for the remainder of the pipeline. The graphics processing strategy for the machine is to perform different image rendering operations concurrently in separate stages of the graphics pipeline. Within each stage of the pipeline, the image rendering task is distributed among multiple processors. The following presentation is based upon [Apg88], [Coo88], [Gel89a], [Gel89b], [Spo88], [Ste88], [Ste89a], [Ste89b], and [Wil88]. In the following presentation, system performance figures are based upon the GS2000 machine.

The Stellar computer system is designed to exploit computational parallelism at the fine-, medium-, and coarse-grained levels. This is accomplished using several techniques.

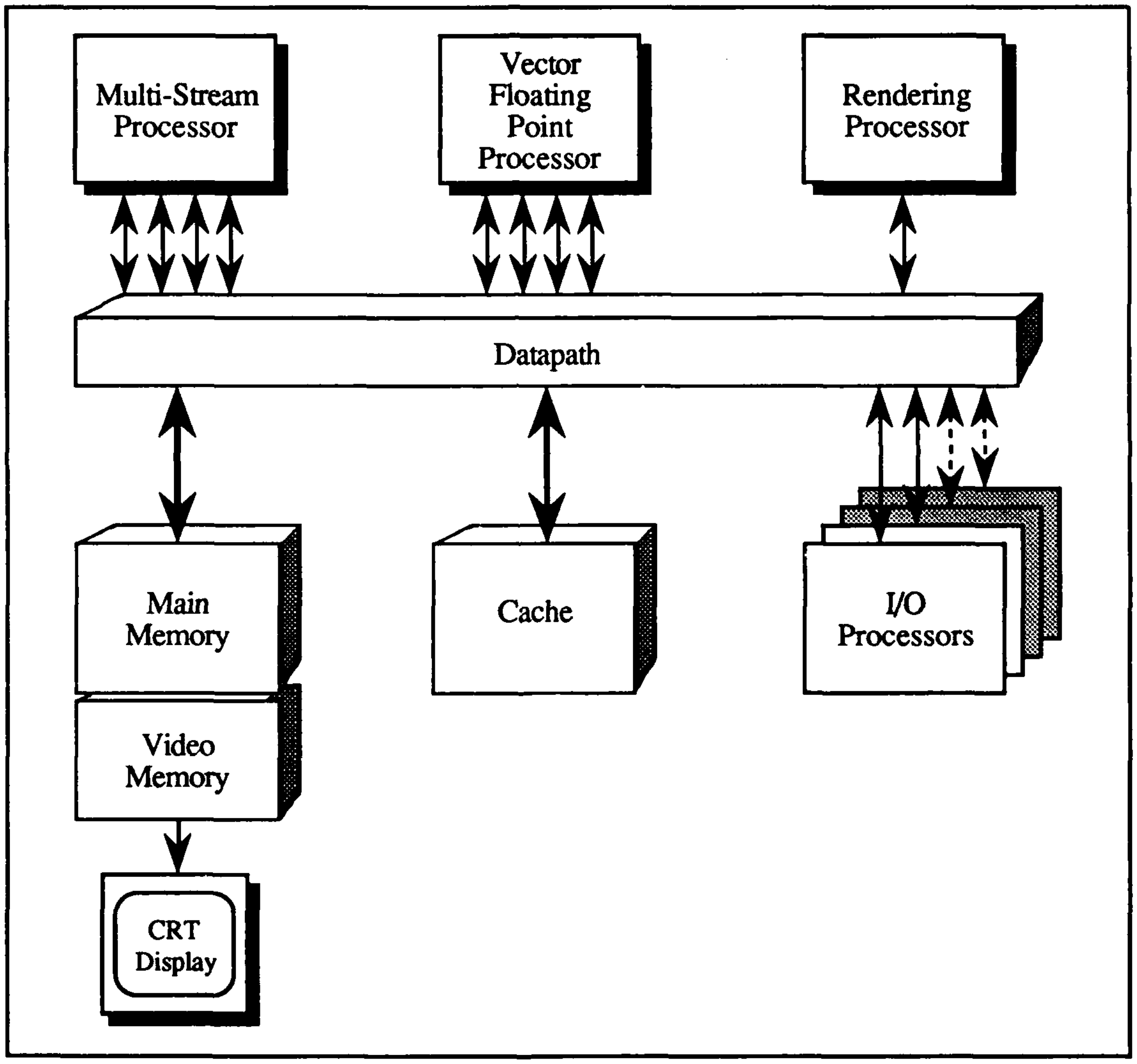


Figure 4.21: Stellar Computer GS Series Architecture (based upon [Ste88]).

The use of independent processors to perform major processing tasks, such as I/O, image rendering, and floating-point calculations, and scheduling of active processes and threads[1] by Stellix™, a Unix™-based operating system, supports coarse-grained parallel operation. Medium-grained parallel operation is achieved through the use of compiler directives to force parallel execution of independent sections of code and the use of executable code explicitly written for concurrent processing. Fine-grained parallel operation is achieved through the use of compiler-detected regions of code which can be executed in parallel or consist of a dependency-free loop. Parallel operation is supported by a shared cache and main memory, threads, compiler directives, concurrency registers, and instructions for inter-thread synchronization. This software architecture for parallel operation is embedded in the hardware architecture in Figure 4.21.

The major components of the Stellar architecture are the Multi-Stream Processor, the Vector Floating Point Processor, the Rendering Processor, the I/O Processors, memory, and the Datapath™. The Datapath is a high-speed switch which interconnects the machine components and supports simultaneous data transfer between all parts of the machine. Except for the concurrency control registers, the Datapath contains all the program accessible registers in the machine. These registers are the independent register sets used by each of the streams in the Multi-Stream Processor, the pixel registers used to send pixel values between main memory and the Rendering Processor, and the 4 I/O registers used to support concurrent I/O by the I/O processors. The Datapath provides the capability for multiple parallel data transfers between machine components over the 512-bit wide buses between memory components and 384-bits wide buses between processing components. This bandwidth supports data transfer rates ranging from 160 MBytes/sec between Datapath and I/O processors to 1.28 GB/sec between cache and Datapath. The Datapath also serves to moderate communication between processors and memory by

[1]The master thread for a process performs serial computations and creates additional threads as needed.

allowing memory access by a processor only during its time-slice, thereby eliminating memory contention issues and a requirement for processor access arbitration. The memory system consists of a cache, translation lookaside buffer, and a main/video memory with a capacity up to 128 Mbytes depending upon configuration. The 1 Mbyte cache is shared by all the streams in the Multi-Stream Processor without synchronization.

The Multi-Stream Processor (MSP) is responsible for performing scalar operations and instruction fetch and decode. The MSP operates at 20 MIPS as a synchronous pipelined multi-processor which supports simultaneous execution on four instruction streams. The MSP pipeline has twelve stages equally allocated among the four streams, as depicted in Figure 4.22.

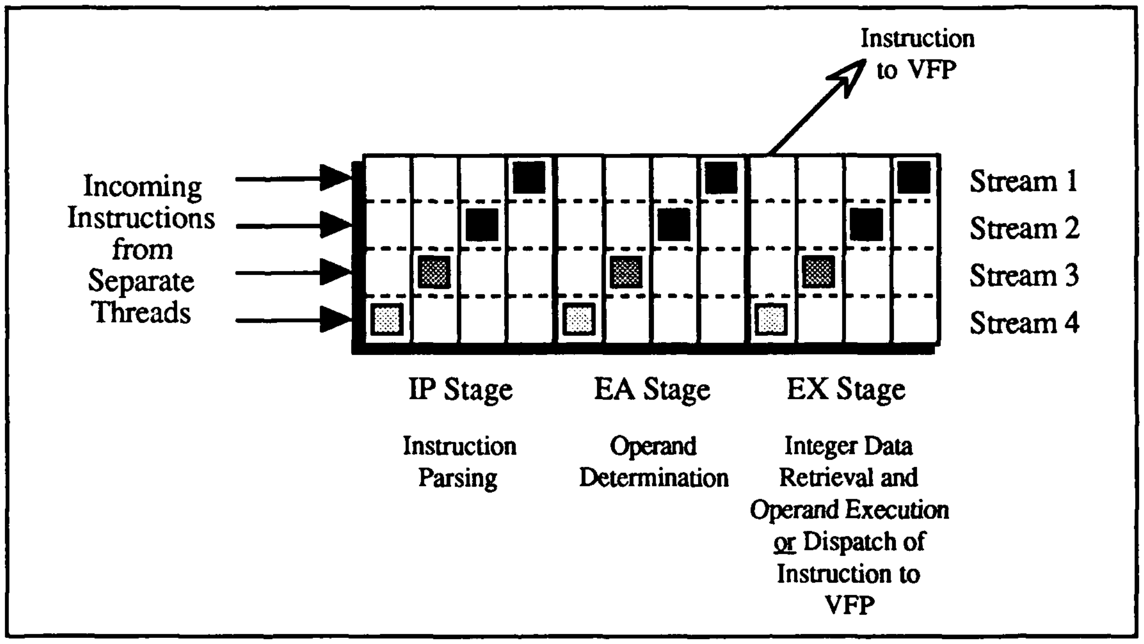

Instruction to VFP
Incoming Instructions from Separate Threads
Stream 1
Stream 2
Stream 3
Stream 4
IP Stage
Instruction Parsing
EA Stage
Operand Determination
EX Stage
Integer Data Retrieval and Operand Execution or Dispatch of Instruction to VFP

Figure 4.22: Multi-Stream Processor Architecture (based upon [Ste88]).

In any given 50 nanosecond clock cycle, each stream has three instructions in the pipeline, with each instruction in a separate stage of execution. The streams can be run as four tightly coupled parallel threads of a single process, as four independent threads, or any other combination of parallel and independent threads as determined by Stellix. A co-processor option adds a fifth stream for scalar-intensive operations. The pipeline is designed to permit each stream to operate independently; therefore, if an instruction in a stream has a cache miss, only that stream is affected and the other streams are not delayed. Independence is achieved by assigning logical processors to a thread, and providing each stream with its own set of integer and control registers and stack. Concurrency between streams is implemented within the MSP using a set of eight dynamically assigned concurrency registers, thereby providing a mechanism for thread synchronization. When a stream in the MSP decodes a floating point instruction, this instruction is dispatched to the Vector Floating Point Processor for execution and the issuing stream waits[1] until instruction completion while the other streams continue processing.

The Vector Floating Point Processor (VFP) is an 80 MFLOPS floating point processor using four or eight independent processors to perform vector and floating point calculations. One or two floating point results are produced by the VFP each clock cycle. The processors are capable of both independent and cooperative action in completing a floating-point instruction dispatched from the MSP, allowing it to simultaneously execute one, two, three, or four vector/floating-point operations depending upon the machine load. To support this independent vector instruction execution, each stream is allocated an independent set of vector registers.

The MSP and VFP serve as the front-end processors for the Stellar graphics pipeline. The full pipeline is shown in Figure 4.23. When operating as a graphics pipeline, the MSP is responsible for traversing the display list for the scene and for cooperating with the VFP to accomplish execution of transformation, clipping, and lighting calculations. As these calculations require floating-point calculation, the MSP serves to set-up the calculation and invokes the VFP to execute the operation. In particular, the VFP performs the calculations required for computing coordinate transformations, clipping

[1]The affected stream does not halt if there are integer instructions or non-dependent vector/floating point instructions available in the stream for execution.

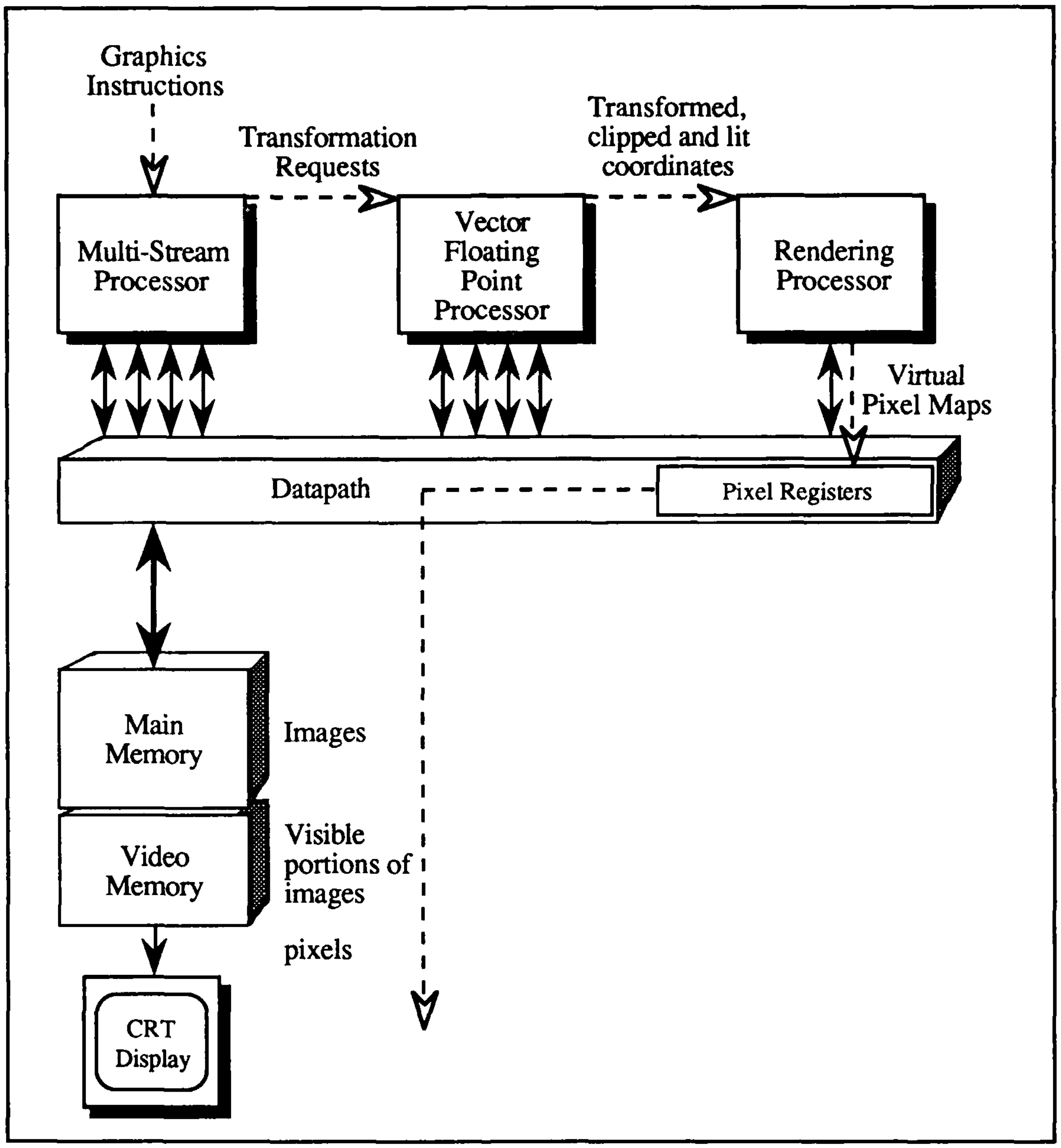


Figure 4.23: Stellar GS Series Graphics Pipeline (based upon [Ste88]).

calculations, generating orthographic and perspective projections and performing lighting calculations. The lighting for each pixel can be calculated for up to 16 colored directional and point light sources with any combination of ambient, diffuse and specular reflection. As each primitive completes processing in the MSP/VFF portion of the pipeline, the result is forwarded to the Rendering Processor where the primitive is rendered according to the lighting and shading techniques that have been specified.

The Rendering Processor (RP) operates at 320 MOPS and uses this speed to compute individual pixel values based on the information sent to it by the VFP. The RP operates on 4 x 4 blocks of pixels when rendering the primitives, this block is moved across the primitive until all points within it have been calculated and written to the Virtual Pixel Map™. The RP performs its image rendering task in three separate stages, the set-up processor, the address processor, and the footprint processor. The set-up processor converts the vertex coordinates and color values for the primitive into the coefficients of linear equations (similar to the Pixel-Planes machine) and forwards this result to the address processor and footprint processor. The address processor is responsible for determining which 4 x 4 blocks in the Virtual Pixel Map are covered by the primitive and for moving the footprint processor across the primitive defined by the linear equations from the set-up processor. The footprint processor is a 16 element[1], SIMD parallel processor which performs the individual pixel lighting calculations, calculates their z-depth, performs required z-buffer comparisons, implements anti-aliasing, performs transparency calculations, and writes the result to a Virtual Pixel Map. The RP is the only component of the graphics pipeline which cannot be altered by the programmer as all the instructions for pixel rendering, such as transparency, anti-aliasing, and texture-mapping operations, are microcoded into the processor itself. Visibility determination is performed on a pixel-by-pixel basis individually by the 16 toes and each toe is responsible for writing its output to the Virtual Pixel Map.

The Virtual Pixel Map is used to implement a 32k x 32k pixel virtual frame-buffer. The images and their associated z-buffer and alpha-buffer[2] information can each be saved as a Virtual Pixel Map for later display or for use in compositing operations using an alpha

[1]The individual processors of the footprint processor are referred to as "toe processors".

[2]The alpha-buffer performs the same functions as the alpha channel in the Pixar machine in that it allows individual frames to be combined through compositing operations which yield new alpha values for each pixel. Unlike the Pixar machine, the alpha values are stored separately from the pixel data.

blend function to determine the relative contribution of each Virtual Pixel Map to the composited image. The maps can be stored in their entirety within main memory, or paged in as required for image operations. The pixel map is displayed by copying the visible portion of the map to video memory.

Stellar relies on the basic speed of its machine to accomplish real-time or photo-realistic volume rendering using cooperating software modules based upon Phigs+ and the X Window system[1]. The three components of the Stellar Application Visualization System (AVS)[2] which perform volume rendering are the the filter, mapper, and renderer modules. The relationships between these modules are depicted in Figure 4.24. These software routines

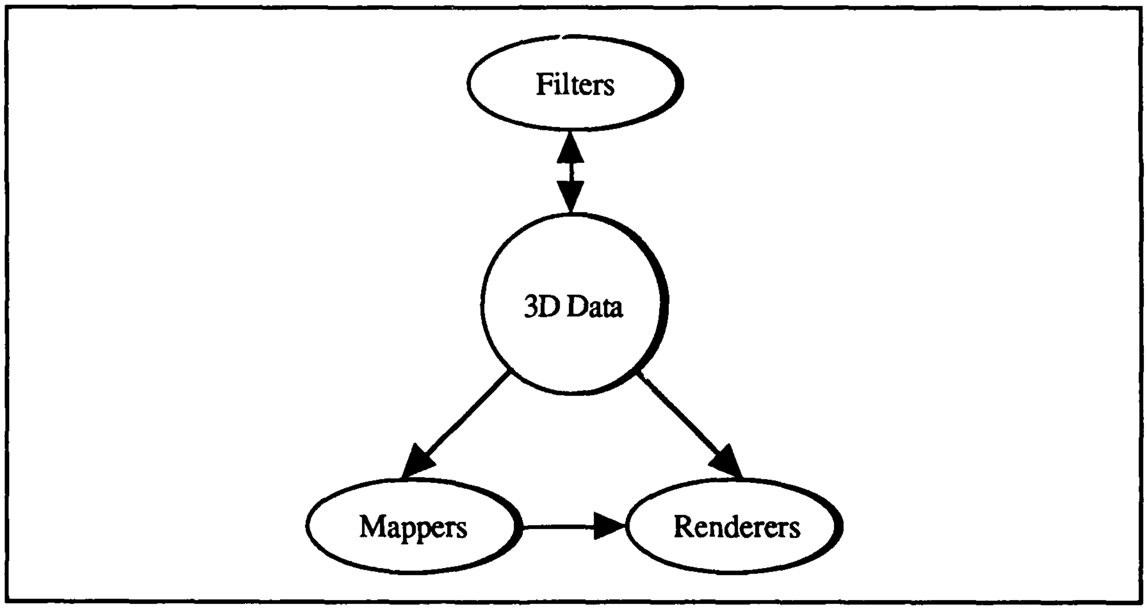


Figure 4.24: Stellar Application Visualization System Architecture (based on [Gel89b]).

[1]The X Window system was developed at MIT to provide desktop management facilities and as a vehicle for building applications. The system is based upon a client-server model and uses interprocess communication to invoke X Window services. A summary of the X Window system is presented in [Sch86].

[2]AVS II was released August 1, 1989 and demonstrated at SIGGRAPH '89 in Boston, Mass.

assume interpolated data, and then manipulate this data to achieve a user specified output. Filters are used to transform input 3D data sets, called data modules, into output 3D data sets. Filter modules provide sampling, pseudo-coloring, contrast modification for feature enhancement, thresholding, and gradient computation. The filter modules can be combined to implement gradient shading, histogramming, filtering, data cropping, data value interpolation, and arithmetic and boolean volume combinations.

Mapper modules are used to transform 3D data sets into 1D, 2D or point data sets which are operated upon by the Rendering Processor, thereby exploiting the pixel writing speed of the Rendering Processor to generate the image. These modules support tiling of a specified surface over the entire volume, detection of a specified surface at a specified level within the volume, selection of an arbitrary slice within the volume, and line-segment displays of vector-field values of selected surfaces. Three renderer modules are supported: a geometry viewer, a volume viewer, and an image viewer. The geometry viewer is the fastest renderer of the three and generates images based on 2D and 3D primitives using the graphics rendering capabilities in graphics library, the Application Visualization System, and the graphics pipeline. The volume viewer is a software based interactive module which uses back-to-front hidden-surface removal for volume rendering and supports thresholding, transparency, and gradient shading. This module uses the MSP and VFP portions of the graphics pipeline with the Rendering Processor functions performed in software (including writing pixel values to Virtual Pixel Maps), thereby increasing the elapsed time to render the image but also increasing the final image quality and available rendering options. The image viewer module generates the highest-quality images of the three modules by scanning voxels directly into screen pixels, allowing the use of multiple scalar fields representing surfaces, opacity, and color to be incorporated into the final image. Like the volume viewer, the image viewer uses the MSP and VFP portions of the graphics pipeline, but does not employ Virtual Pixel Maps. The filter, mapper, and renderer modules cooperate in the image formation process with demand-driven requests for data to earlier modules in the pipeline and data-driven processing in a requesting module when the data becomes available.

4.4.4.1 Shading, Anti-Aliasing and Hidden-Surface Removal Techniques.

The Stellar computer performs most of its high-level graphics processing in software, but the shading, anti-aliasing, and hidden-surface removal operations on individual pixels are accomplished in hardware within the Rendering Processor. The Rendering Processor supports Phong, Gouraud, and distance shading as well as texture mapping to provide a sense of depth to the image. Gradient shading is implemented within the filter module volume visualization system of the Stellar computer. Anti-aliasing is performed by using a blend function. This function uses a 3 x 3 filter and performs 9 renderings of each image. The coefficient of a filter element is multiplied by the rendered pixel value to determine the final pixel value for each rendering. The nine images are then composed to yield one final, anti-aliased image. As this process takes too long to permit real-time interaction with the image, anti-aliasing is performed only when the user is not interacting with the display (ie. when image motion ceases). The Rendering Processor performs hidden-surface removal operations on a pixel-level using the z-buffer algorithm. To accomplish hidden-surface removal within volumes, two software options are available. The volume viewer module performs back-to-front hidden-surface removal. The highest-quality images employ the cell-by-cell pixel scanning approach implemented in the image viewer module.

4.4.4.2 Degree of Parallelism and Machine Performance.

The Stellar computer is a highly-parallel mix of MIMD and SIMD processors in one machine. I/O, volume rendering, pixel-level calculations, and image display are all overlapped operations using a variety of independent processors and pipelines supported by a switch which allows all the computational elements to communicate at high speed with little contention. The graphics pipeline is capable of rendering 800,000 points/second and transforming and rendering 150,000 pseudo-colored, z-buffered, Gouraud shaded 100-pixel polygons/second. The machine is capable of performing the volume rendering operations required by medical imaging applications at a resolution of 1280 x 1024 pixels, but performance figures have not been published.

4.4.5 Sun™ TAAC-1™ Application Accelerator.

The Sun™ TAAC-1™ Application Accelerator is an add-on set of boards for Sun workstations designed to perform compute intensive calculations as well as the graphics processing operations required to render image displays. The system is based upon a processing unit comprised of multiple buses and non-pipelined calculating units. The buses and computing units can be configured into various software pipelines to provide the TAAC-1 its 30 MIPS peak performance. Like the Ardent and Stellar machines, the TAAC-1 refrains from embedding graphics algorithms in hardware but relies upon the performance of the general purpose hardware to provide high-speed image renderings. This approach affords flexibility in both the configuration of the software graphics pipeline and in the choice of algorithms implemented to render images. The following presentation is based upon material in [Aus88], [Joh89], [Sun88a], [Sun88b], [Sun88c], [Sun88d], [Sun88e], and [Sun88f].

The goal of the TAAC-1 design is to provide a self-contained graphics accelerator which can perform the integer and floating point calculations required for image processing and 3D renderings at near real-time rates. The processing strategy consists of using a flexible very long instruction word (VLIW) architecture, multiple memories, processors, and buses. Hardware support for 2D and 3D array addressing coupled with software-embedded algorithms help attain this goal. The use of VLIW instructions allows for multiple interconnection schemes between the independently operating components of the processor unit. The use of multiple memory banks, processors, and buses allows the TAAC-1 to maintain a steady flow of operands to the computational units with little contention between them for bus access. The capability for direct addressing of 2D and 3D arrays reduces the number of calculations performed when accessing these arrays. These hardware elements provide sufficient computing power to allow graphics algorithms to be coded in software instead of hardware and still provide a rapid image generation capability. The processing strategy is realized in the hardware architecture depicted in Figure 4.25.

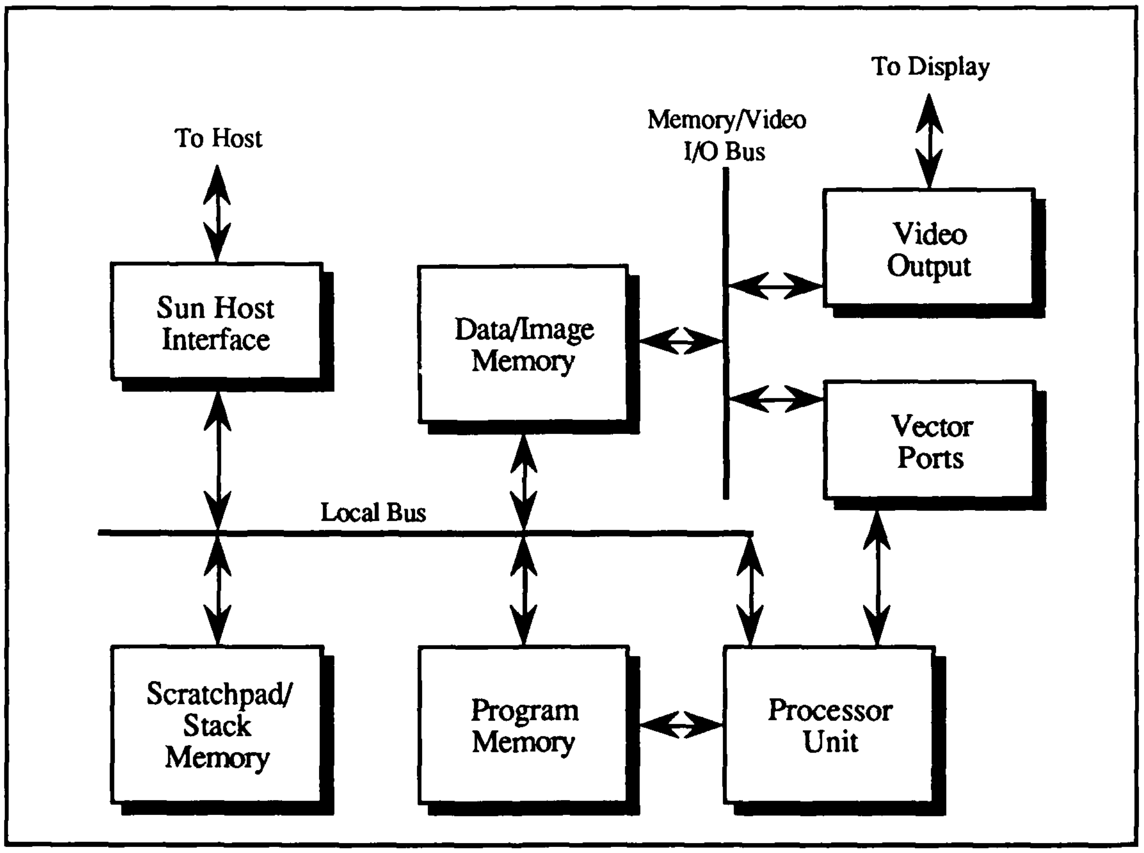


Figure 4.25: Sun TAAC-1 Application Accelerator Architecture (based upon [Sun88d]).

The major components of the TAAC-1 are the Sun-3™ or Sun-4™ host workstation, the host interface, the 8 Mbyte data/image memory, the program memory, the scratchpad/stack memory, the vector ports, the video output unit, and the processor unit along with the five 32-bit buses. The host is responsible for providing disk I/O services, the user interface, libraries, and window management for the TAAC-1. The host interface unit manages the mapping of TAAC-1 bus address space into the bus address space of the host. Because the TAAC-1 uses 200-bit instruction words for control of buses and processors, program memory is organized as a 16k x 200-bit memory and given a dedicated bus for processor unit access. The vector ports are dual 32-bit buses operating at 50 Mbytes/second connecting the processor and the serial ports of the data/image memory banks. These buses allow the processor to access one bank of the two-bank data/image memory through one serial port while the video output unit simultaneously accesses the other bank through the other port. The video output unit supports four independent 8-bit channels for color, one of which can be used as a single 8-bit alpha overlay channel, each with independent 8-bit in, 24-bit out look-up tables, allowing the TAAC-1 to display 24-bit true color or 8-bit pseudocolor. Video resolution is software configurable to either 1024 x 1024 or 640 x 480 resolution.

The data/image memory is organized as 2 million 32-bit words in VRAM which can be used to hold images or data. When the memory is used to store images, the dual-access capability provided by the vector ports allows one bank to be addressed vectorially as a 1024 x 1024 frame buffer while the other bank is updated. In addition to vector access through the vector ports, random access is provided using the random access ports on the local bus. The data/image memory is designed to support 1D, 2D and 3D array access, using a special purpose access processor register, without calculation of strides or offsets. The memory is organized as eight 1 Mbyte sectors, with each sector having four blocks of 64k words. For 1D addressing, the 8 Mbytes of memory are treated as sequential memory locations, with consecutive x-array indices located in consecutive blocks, that is, block 0 contains x-array indices 0, 4, 8, etc. For 2D addressing, the memory is treated as though it were a 1024 x 2048 image array. The 2D address is formed by bit-shuffling in hardware so that the blocks in a sector are treated as though consecutive x-array indices lay within the same block and not in consecutive blocks. The two modes of 3D addressing, slice and dice, provide different access methods. Dice addressing supports cubes ranging in size from 8 x 8 x 8 to 128 x 128 x 128. Addressing for this mode is performed by rearranging the seven bits for each of the x-, y-, and z- array indices so that the memory address is composed of alternating x, y, and z index bits. In slice mode addressing, the memory is treated as 32 slices of 256 x 256 data. The slice number is selected by using the z address to select the sector and block, with the x- and y- array indices used to select the x- and y-coordinate value in the slice as in 2D addressing.

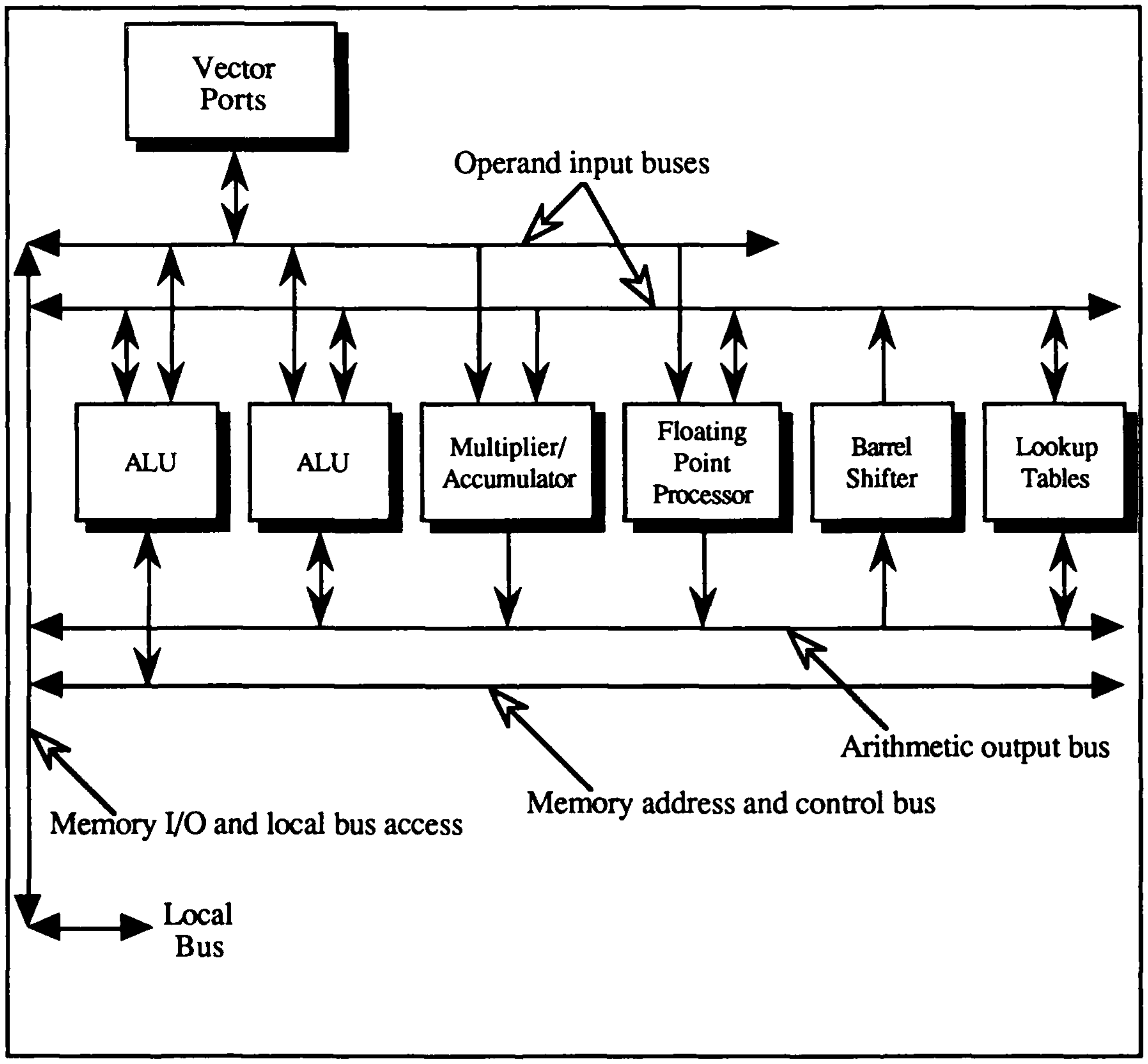


Figure 4.26: Sun TAAC-1 Processor (based upon [Sun88d]).

The Sun TAAC-1 processor is depicted in Figure 4.26 The peak processor performance of 30 MIPS is attained when simultaneous integer operations are being performed at the ALUs, the multiplier/accumulator, and the barrel shifter. The floating-point unit is designed to perform one multiply and one add each cycle, giving it a peak performance rate of 12.5 MFlops when the vector ports are used as the data source. One of the look-up tables is committed to providing seed values to Newton-Raphson iteration for reciprocal and square root computations. The other, user-programmable, table is used for image processing and image histogramming. On an instructon-by-instruction basis, these elements of the processor can be reconfigured into processing pipelines using the buses to provide inter-stage communication. Normally, one ALU is dedicated to address calculation with the other ALU employed for data processing. The barrel shifter and multiplier/accumulator can then be combined with the data processing ALU to perform pipelined operations on images or data. An additional register in the processor provides a hardware-based z-buffer hidden-surface removal capability but its use limits the display to 16 bits/pixel for gray-scale and color instead of the normal 32-bits.

The graphics packages provided with the TAAC-1 provide a basic capability for image rendering, with the option of modifying the routines to incorporate new processing algorithms. Medical image processing can be performed using polygon-based surface-presentation techniques, any of three volume rendering tools, Cloudvu, Rayvu, and Cubevu, or the TAAC-1 image processing library. Given a 2D surface-contour, the visible surface can be rendered at the rate of 5,500 24-bit color polygons per second. Cloudvu is a volume rendering program which operates on point primitives as defined by three-dimensional coordinates, a surface normal, and an 8-bit color value. This tool can render 225K z-buffered points/second using depth shading. Cubevu is employed as a high-speed editor for a volume, allowing for the rapid display of slices of the volume from arbitrary orientations and with arbitrary cutting planes. The volume is modeled as a solid cube, thereby eliminating the capability for depth renderings of the volume. This volume model simplifies the computation task, so that the visible surface of a slice can be rendered in real-time using nearest-neighbor or trilinear interpolation to determine pixel values from the 8-

bit or 16-bit voxel data. The volume editing capability is provided using one "flying" slicing plane which can be placed at an arbitrary orientation to the three axes and six additional planes, each placed perpendicular to either the x-, y-, or z-axis. These six planes are limited to back and forth motion along their axis. Once an individual slice is extracted from the volume, it can be used for additional 2D processing, such as adaptive histogram equalization. The volume of interest defined by the cutting planes can be dispatched to the Rayvu program for rendering as a full-depth 3D image. Rayvu is the ray tracing tool provided for the TAAC-1 which is used for compositing and ray tracing of volumes edited by the Cubevu package. Image composition is performed by classification of up to eight substances by density value, with an associated color and opacity for each substance used to render shaded, transparent surfaces. As in Cloudvu, pixel values are determined using nearest-neighbor or trilinear interpolation on the voxel data, with a surface normal approximated by a central difference gradient.

<u>4.4.5.1 Shading, Anti-Aliasing, and Hidden-Surface Removal Techniques.</u>

When employing the Cloudvu package, flat or Gouraud shading models can be employed. The Rayvu package supports image rendering using either the sum of densities along each ray, a weighted density average, or the largest density value as the shading model. Hidden-surface removal can be performed using the hardware-based Z-buffer or via user created software. Anti-aliasing, if performed, is accomplished using trilinear interpolation in both the Cubevu and Rayvu packages. If the TAACART toolkit is installed, then anti-aliasing can be performed using stochastic sampling. As this system places its image processing operations in software, the user has the option of extending the capabilities of the machine by constructing additional image rendering tools.

<u>4.4.5.2 Degree of Parallelism and Machine Performance.</u>

The Sun TAAC-1 is a moderately parallel machine due to its capability for operation as a logical software pipeline on an instruction-by-instruction basis. The TAAC-1 owes its image generation speed to the speed of its floating-point unit and its 2D and 3D addressing scheme. The Rayvu package can render a 256 x 256 x 32 volume using trilinear interpolation to generate pixel color and transparency effects in approximately two minutes. The Cubevu package can display the surface of an arbitrary 2D slice from a 256 x 256 x 256 cube in .039 seconds.

4.4.6 Section Summary.

The commercial machines reflect a narrower range of approaches to the medical image rendering problem than is the case for the research machines. The machines are all basically pipelines with some form of hardware assist at the pixel generation level. A significant aspect of all these machines is their common decision to place the image rendering algorithms in software rather than hardware, thus providing a capability for future enhancements to their rendering algorithms. The effect of this decision is reflected in the performance and hardware of the machines. The machines either take a considerable amount of time (minutes) to generate their images (*viz* Pixar) or employ special purpose hardware in the image generation pipeline to attain acceptable, but not real-time, performance (*viz* Stellar), with the other machines falling between these two extremes.

The Pixar machines reflect the low end of this scale, with their "pipeline" existing only in a logical sense, as it is implemented through the use of multiple passes over the data set by the same processors with the results of each pass composited into a final image. The hardware assist for pixel generation is limited to its dual-ported frame-buffer which allows direct access to the video image by both the Chap and the video display generator. This machine is significant for its demonstration of image compositing techniques for medical imaging and its resulting high-quality images. The Sun TAAC-1 limits its "pipeline" to instruction-commanded reconfiguration of the processing unit. The TAAC-1 demonstrates the benefits of clever algorithms, as its performance derives from its ability to access 3D arrays quickly and the use of scene-editing to limit the amount of time required to achieve the desired volume orientation before rendering the full image. The Pixel Machine provides both a multistage pipeline and dedicated-hardware support for pixel generation, resulting in a significant performance improvement over the TAAC-1 and Pixar efforts. Its performance is limited by the narrowness of its pipeline, and its performance when rendering voxel-based images undoubtedly suffers from this. The Ardent and Stellar machines, with their dedicated hardware for pixel generation and multistage pipelines, provide a somewhat wider image rendering pipeline than the Pixel Machine with another significant performance improvement. While Stellar has not yet published medical image rendering times for voxel-based data, its basic similarity to the Titan and wider use of special-purpose hardware suggests that it should be capable of equivalent performance. As of this writing, all these machines suffer from a paucity of software for medical imaging purposes, and none has achieved real-time performance in the rendering of high-quality images.

4.5 Summary.

In the preceding pages I presented various solutions to the problems of displaying medical image data. These solutions ranged from the rather straightforward (i.e., the MIPG machine, which uses insightful algorithms on the computer provided with the CT scanner equipment) to the very specialized (the Pixel Planes machine and the Voxel Processor). In all cases, certain tradeoffs were evident. In particular, note that the general purpose solutions are either slow or they use very expensive equipment (Farrell's solution), while the special purpose solutions are quite powerful, but they are geared to only one problem. If there is not full utilization of the time on these special purpose machines, their cost effectiveness is diminished. The appearance of architectures like the Titan, GS2000, and

AT&T Pixel Machine reflect the realization of the need to provide a high capability graphics rendering machine within a general-purpose architecture. In these machines, the twin goals of photo-realism and real-time performance have not been met, but they do make large strides towards the attainment of the imaging environment described at the beginning of this chapter.

It behooves us to ask what can be learned from these attempts and then couple these lessons with the state of current technology to provide general purpose medical imaging solutions that can have the effectiveness of the currently available special purpose ones.

First, the special purpose solutions indicate the right approach - parallel computation, on as massive level as possible, and in a grain that is amenable to utilization of the hardware by other tasks. These other tasks should be of interest to medical care providers, as these medical imaging machines have to be situated in the hospitals and clinics, at least till such time when ultra high-speed communication media will be available. This implies that these parallel machine solutions have to be achieved by fairly general purpose computers.

Second, the basic data for the display should be the actual slice data, possibly with interpolated data added. Whatever additional processing may be necessary should be done as requested by the user. These are controversial conclusions, and can be supported only when proof is offered that the additional pre-processing can be done in real time.

Third, the need for new approaches to the medical imaging task is still with us. For the foreseeable future, hardware can not provide the processing speed required to generate photo-realistic images in real-time. Insightful algorithms and concepts, such as the Sun TAAC-1 3D memory addressing scheme and Kaufman's cube memory, which reduce the computational burden, are required, especially as the medical imaging community is now acquiring the capability for even higher resolution data acquisition.

Last, whatever special-purpose hardware support is necessary to create a real time environment, it should be in the form of co-processors or special purpose display equipment. This additional hardware should be separated relatively cleanly from the actual general purpose processor(s) within the overall image rendering pipeline architecture. This approach, taken in the Ardent, AT&T, Pxpl5, and Stellar machines, combines image rendering flexibility with image rendering speed. At this time; rapid, realistic shading is possible only with the use of dedicated shading hardware, but because shading is normally accomplished at the end of the image processing pipeline this hardware should be entirely devoted to shading. The front-end of the pipeline is responsible for the object space to screen space projection. Because of the intense research effort concentrated on this portion of the image rendering task, it is not advisable to lock-in the algorithms employed by embedding them in special-purpose hardware. Instead, a flexible front-end allows for ready incorporation of innovations in rendering as well as providing additional options for rendering within a single pipeline using known approaches.

This concludes the survey of previous designs of 3D medical image rendering machines. The next chapter presents the initial results of my research into a commercial parallel-processor based 3D medical imaging environment.

CHAPTER V

MEDICAL IMAGING PIPELINE PERFORMANCE AND MULTIPROCESSOR INTERCONNECTION TOPOLOGY CONSIDERATIONS

5.1 Introduction.

This chapter presents the initial results of my investigation into processing algorithms and parallel processing host architectures. This investigation is carried out within an imaging pipeline architecture derived from the parallel-processing primitives first described in [Rey85] using a MIMD[1] parallel processing approach within each pipeline stage. Two parallel processing interconnection topologies are examined, the mesh and hypercube, within a simulated multiprocessor machine. Using these topologies, a range of message packet sizes and image volume sizes are investigated and the performance of these topologies across the range of packet sizes is discussed. As is shown, communication costs, and hence interconnection topology, are minor performance considerations. However, this pipeline performance analysis demonstrates that the pipeline stage wherein hidden-surface removal is accomplished is the pipeline performance bottleneck. Therefore, the hidden-surface removal operation is investigated in depth in this chapter and in Chapter VI. The discussion in the present chapter of recursive hidden-surface removal is designed

[1]Multiple-Instruction, Multiple-Data

to establish upper- and lower-bound performance for the pipeline and to establish the pipeline stage wherein recursive hidden-surface removal is performed. These limits are established using two different variations of two different implementations of back-to-front and front-to-back recursive hidden-surface removal algorithms. With these results in hand, conclusions about packet size and host multiprocessor interconnection topology are made. These performance bounds, and the definition presented in this chapter, lead into the exposition in Chapter VI of a new algorithm for performing recursive hidden-surface removal which renders a three-dimensional (3D) image approximately 50% faster than previous recursive hidden-surface removal algorithms. The performance of the new recursive hidden-surface removal algorithm within the simulated multiprocessor architecture using the recommended interconnection topology and packet size is discussed in Chapter VI.

Because the hidden-surface removal operation is critical to pipeline performance, new implementations of the back-to-front and front-to-back hidden-surface removal algorithms were developed. This chapter introduces this modified recursive hidden-surface removal (HSR) algorithm, the editing recursive HSR algorithm, which is designed to rapidly perform hidden-surface removal and is suitable for scene editing purposes. The use of this algorithm within the multiprocessor interconnection topologies results in a significant reduction in the time spent in the Subscene Generator stage of the medical image processing pipeline. This reduction, in turn, results in a substantial overall decrease in image rendering time required by the medical imaging pipeline within either the mesh or the hypercube multiprocessor interconnection topology.

This chapter is organized as follows. Section two contains basic information concerning the simulated multiprocessor environment; the data model, the assumed coordinate systems, a description of the image processing pipeline, the operations performed in each of the stages of the pipeline, and the pipeline image correctness and timing methodologies. Section three describes the multiprocessor interconnection topologies which host the pipeline and the pipeline - processor node assignments and the simulated multiprocessor. Section four contains image processing pipeline performance results for each interconnection topology across various packet sizes. In preparation for the description of the editing recursive hidden-surface algorithms described in section six, section five presents a definition of recursive hidden-surface removal algorithms. Section six describes the editing back-to-front and front-to-back recursive hidden-surface removal algorithms. Section seven contains pipeline performance results obtained using the algorithms described in section six. Section eight presents a comparative analysis of the performance of the interconnection topologies and conclusions as to which topology is the "best" choice and an appropriate packet size. Section nine presents general conclusions for the chapter.

5.2 Data Model, Coordinate Systems, and Image Processing Pipeline.

This section describes the data model, assumed coordinate system and pipeline operation within the simulated multiprocessor (described in Section 5.3). In addition to describing the general flow of data through the image processing pipeline, the operation of each stage is described and the methodology employed to time the operation of the pipeline is discussed.

5.2.1 Data Model and Coordinate Systems.

In order to obtain high image resolution, I chose to employ the voxel data model for final 3D object display. The voxel model was selected because: 1) the voxel data model avoids the computational expense of generating new surfaces/contours in response to user inputs which change the visible portion of the scene; 2) the voxel model avoids introducing artifacts into the image which arise from classification decisions concerning the presence of a given surface within each voxel; 3) the data is readily partitioned among the nodes of either interconnection topology. To simplify the presentation of the thesis, when referring to voxel data the term density is taken to mean the image data produced by Computerized Tomography (CT), Magnetic Resonance Imaging (MRI), ultrasound, Single Photon Emission Computed Tomography (SPECT), or Positron Emission Tomography (PET) imaging modalities.

The origin for object space and image space is assumed to lay at the front, lower left of each volume. Object space coordinates are specified by the unprimed characters: x, y, z and image space coordinates are specified by x', y', and z'. When required for presentation clarity, screen space coordinates are represented by u and v, otherwise they are assumed to correspond to the x' and y' image space coordinates.

5.2.2 Image Processing Pipeline Operation.

The general processing strategy I adopted for the image processing pipeline is based on Farrell's strategy of dividing image processing between architectures suited to each phase of processing. Therefore, the image generating operations are partitioned between a host (or node(s) of a multiprocessor) and the pipeline. The computations are allocated so that communication costs between the host and pipeline are minimized while at the same time allocating the partitionable computationally intensive operations to the pipeline. The attached host (or dedicated multiprocessor node(s)) has responsibility for transmitting image data to the pipeline, gathering user inputs, receiving processed image data back from the pipeline, and for driving the graphics display. For simplicity of exposition, the input and output operations are assigned to separate hosts in the following description. The image processing pipeline described below is based on the parallel processing primitives presented in [Rey85]. The pipeline consists of several stages which operates on a three-dimensional scene formed by interpolation of CT, SPECT, PET, MRI, or ultrasound image data. Figure 5.1 depicts the full-scale image processing pipeline.

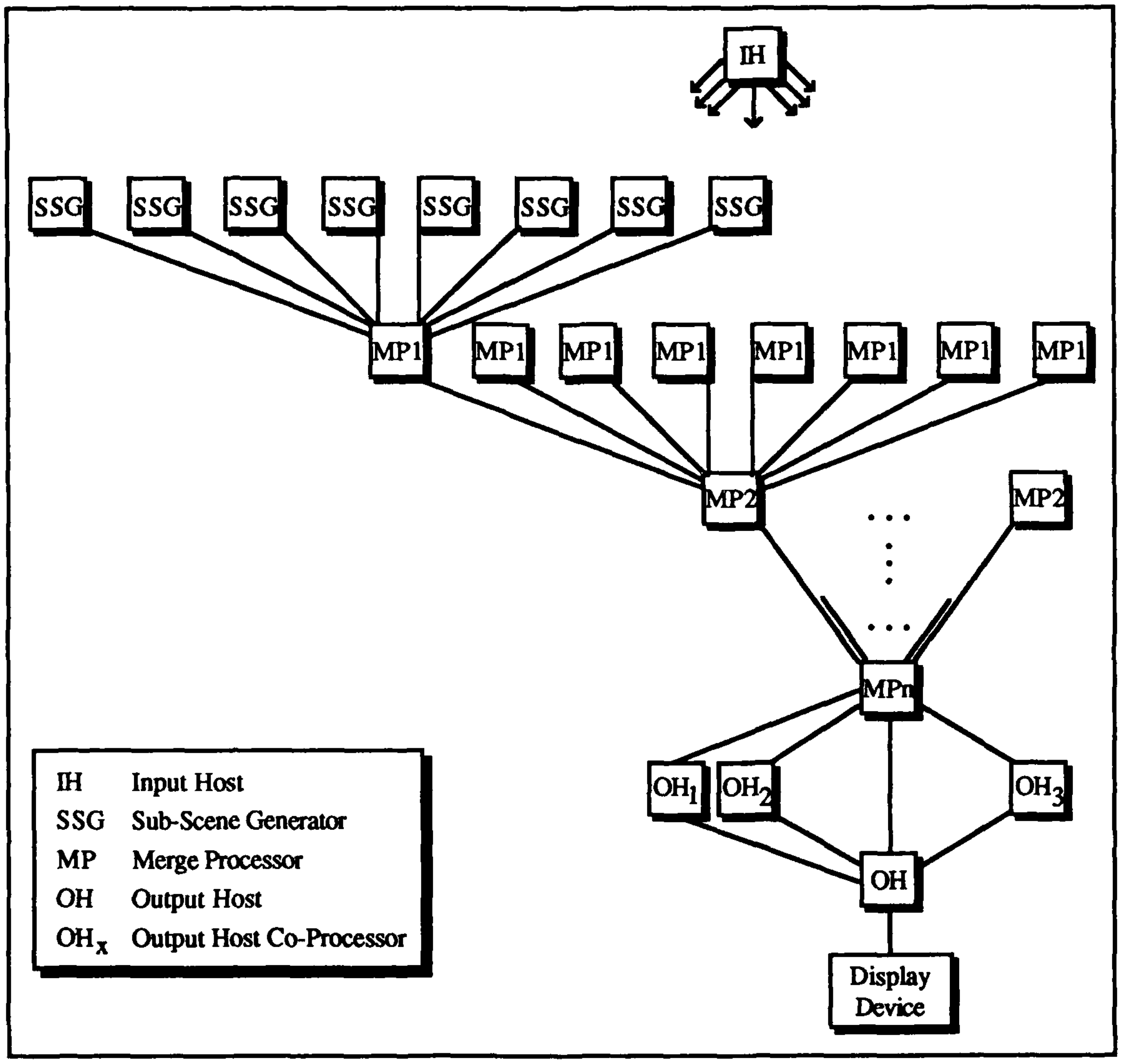


Figure 5.1: Full-scale Image Processing Pipeline.

The operational assignments for each stage of the pipeline are as follows. The Input Host (IH) has two functions. Its primary function is to gather user scene editing inputs and broadcast them to the remaining processors in the pipeline. User inputs consist of commands specifying cutting plane location, translation, density window, image zoom, and rotation. The IH also performs the initial distribution of the integer voxel values to the

Sub-Scene Generator (SSG) processors. This data distribution equally partitions the volume among the Sub-Scene Generators, with each SSG receiving a continuous, discrete volume for it to operate upon. Each Sub-Scene Generator is responsible for performing anti-aliasing, scene rotation, hidden-surface removal (HSR), and object-space-to-image-space mapping on its subset of the volume based on the user inputs broadcast by the Input Host. Anti-aliasing is performed using supersampling to double resolution with resampling before output to the next stage. Because HSR, rotation, and object-space-to-image-space mapping are computationally intensive operations and must be re-accomplished for each change in scene orientation, a combined approach to the these operations should be employed. Recursive hidden-surface removal algorithms are a combined solution because they perform scene rotation and object-space-to-image-space mapping in conjunction with HSR. Therefore, the SSGs perform hidden-surface removal using either a back-to-front or front-to-back recursive hidden-surface removal algorithm. The pseudocode for the basic Back-to-Front (BTF) and Front-to-Back (FTB) hidden-surface removal algorithms used in the SSG stage is contained in Appendix C.

When each SSG finishes processing its volume of voxels, the output 2D scene, consisting of an integer density value and an integer z value for each coordinate, is transmitted to its assigned Merge Processor in the next stage. Each Merge Processor (MP) gathers the output of the eight SSGs which send it input and merges the eight 2D input scenes to form a larger 2D output scene. Successive stages of MPs are used to construct ever larger subsets of the complete image. The final pipeline stage, consisting of one MP, merges the eight scenes from the eight MPs in the previous stage into the final, full-scale scene. The output from the last MP stage is sent to the Output Host (OH) stage for 8-bit gray-scale assignment, image magnification (zoom), and shading to provide a three-dimensional effect. To speed up the operation of the last stage, the MP's output is equally divided among the Output Host and Output Host$_1$, Output Host$_2$, and Output Host$_3$ for processing, before final display of the image by the Output Host. Shading is performed in software, even though image shading hardware exists, to demonstrate that the pipeline operates correctly within the simulated multiprocessor.

5.2.3 Individual Stage Operations in the Pipeline.

This sub-section amplifies the pipeline stage description in the preceding section by providing implementation details concerning the operation of each stage. The IH is responsible for acquiring user inputs and distributing them to the remaining nodes of the pipeline as well as for volume sub-division among the SSGs. The goal of the volume partitioning step is to distribute the image processing workload equally among the nodes in a given stage via distribution of equal-sized volume subsets among the SSGs. Given a set of slices produced by a medical imaging modality, an N x N x N volume can be formed by linear interpolation of the input slices. The resulting cube-shaped volume is partitioned octant-wise equally among the SSGs so that for P SSGs, each SSG receives a continuous, discrete $N/\sqrt[3]{P} \times N/\sqrt[3]{P} \times N/\sqrt[3]{P}$ volume[1] of voxels. The data distribution is further constrained by the requirement that the union of the volumes in the eight SSGs reporting to a single MP forms a single suboctant in object space. As soon as the volume for each SSG is extracted from the cube, the data in the volume is dispatched to the SSG in a message packet.

The scene display information is distributed to pipeline nodes using two separate message packets. The editing packet contains the user instructions for cutting plane location, translation, and threshold window settings. The other, scene orientation, packet contains the Sequence Control Table (SCT) used by the MPs and SSGs to perform image rendering. The SCT contains eight entries, one for each scene octant, with each entry

[1]N and P must be a power of 2 in magnitude.

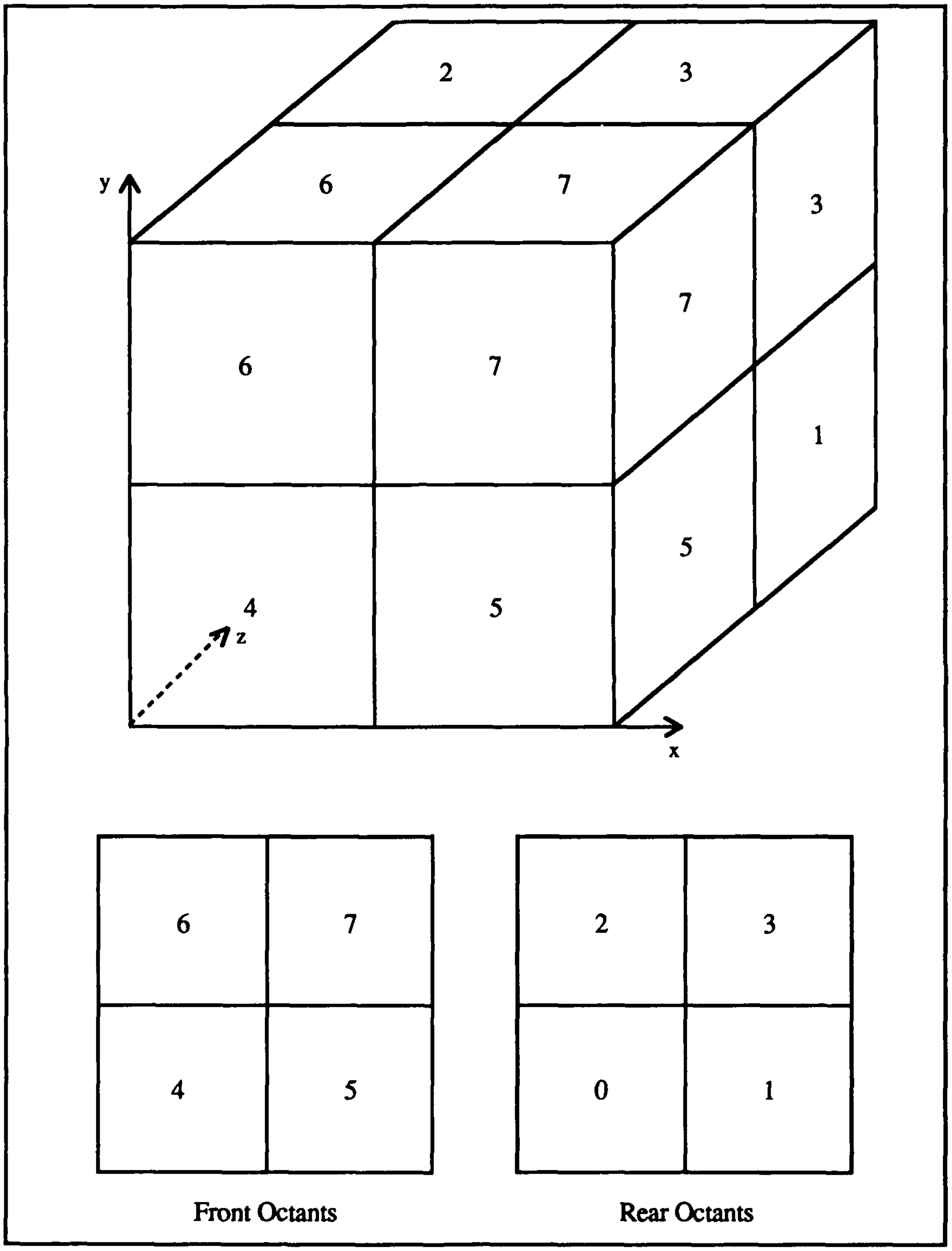


Figure 5.2: Octant Numbering Scheme.

consisting of an object space scene octant number and the octant generator coordinate[1] for this scene octant after it has been transformed into image space. Octant numbers are assigned according to the scheme depicted in Figure 3.7 (repeated below as Figure 5.2). The coordinates in the SCT are left-shifted to scale them sufficiently for the recursive hidden-surface removal calculations at the SSGs, and sorted into either back-to-front or front-to-back order depending on the type of algorithm being run at the SSGs and MPs.

The SSGs are responsible for supersampling to double resolution the suboctant data received from the IH and storing it in a 3D array as well calculating the object space coordinates used to generate the remaining points in the object space suboctant before acting upon the first user input. Upon receipt of the orientation packet, each SSG uses this information to compute the image space octant coordinate which corresponds to its object space octant generator coordinate. SCT values are then scaled down from object space size to the SSG volume size by right-shifting each coordinate entry once for each MP stage in the pipeline. At this point, each SSG has enough information in the form of object and image space coordinates and image space offset values to perform recursive HSR as described in Chapter 3 and Appendix C. When the HSR procedure concludes, the SSG has stored the visible voxels' density values and z' coordinates in a two-dimensional (2D) array. The density values are resampled to normal resolution using a 4 x 4 filter and the corresponding z' coordinates are averaged over the same 4 x 4 area before transmission of the output data to its MP for merging with its sibling SSGs' output.

The MPs take the output from the preceding stage and merge it in either back-to-front or front-to-back order as defined by the SCT. Merging is a two-step process. First, the MP determines the image space suboctant each input derives from. Second, each input

[1]The generator coordinate can lie at either the front-lower left (FLL) or the subcube center (SCC) of the octant.

is selected in SCT order and laid down in a 2D output array. When the last input is processed, it is sent to the next stage.

The OH stage is responsible for gray-scale assignment, zoom, and image shading. Gray-scale assignment is performed using table look-up. For timing purposes, only gray-scale assignment is performed on the density values produced by the SSGs, even though the z' data required for shading is available. The decision to forego timing the shading operation was made because the computational cost of shading and its position in the pipeline indicate that this operation should be performed using dedicated shading hardware[1]. For image-display, distance-only shading is performed using the z' coordinate values produced by the SSGs using a single light source co-located with the observer. This shading method is used because it provides a 3D effect without introducing additional lighting effects which could mask defects in rendered images. However, the z' information captured during the rendering process allows the gradient shading algorithm to be used to improve rendered image quality. Gathering additional information during the rendering process is possible, but comes at the cost of increasing the computation and communication processing.

5.2.4 Image Correctness Assessment and Pipeline Timing Methodology.

Because of difficulties in obtaining consistent timing results within the HP environment, a two phase demonstration-timing approach is used to assess pipeline performance. The demonstration phase consists of proving the accuracy of the algorithms used in each stage by running the pipeline within each of the simulated connection

[1]As in the Ardent Titan, Stellar GS1000 and GS2000, Pixel Planes machines, Voxel Processor, and AT&T Pixel Machines.

topologies described in section 5.3. Pipeline operation results in the production of actual images which are verified for correctness in two separate steps. First, images produced by the pipeline are visually inspected for errors. Second, the image space coordinates produced by the algorithm employed in the SSG stage are compared with the image space coordinates computed using rotation and translation matrices to perform the object-space-to-image-space transformation. The algorithm used in the SSG is deemed accurate if its image space coordinates are within $\pm 10^{-4}$ of those produced by the matrix method.

The timing phase, begun after the accuracy of the image formation algorithm is confirmed, consists of gathering timing results for each of the pipeline stages by either running its code in a stand-alone environment for different image sizes or during normal pipeline operatio[illegible] [illegible]cause the IH and OH are the only processors active within their respective st[illegible] during pipeline operation, timing figures for these stages can be gathered during pipeline operation. The SSG and MP stages are timed separately from the pipeline to attain consistent throughput figures for these stages and to gather results for larger image sizes. Timing results are obtained using the timing calls available in HPUX™, with the high, low, and mean production time recorded along with the standard deviation for each stage[1]. The process timing results reported in this work should be interpreted as relative measures of algorithmic performance. Actual implementations would not have the HPUX overhead to contend with, which could alter the magnitude, but not the relative performance, of the timing results for each stage of the pipeline.

Internode communication time is computed based on the number of data bytes and amount of packet header information moved between nodes and the number of node-to-node hops each data packet must make before reaching its final destination. Internode

[1]The timing results reported in this work have a standard deviation of one tick or less, where one tick of the system clock is 1/50th of a second. The standard deviation, low, and high times do not affect the results or alter the conclusions presented in this work, and so are not reported.

communication times are gathered during normal pipeline operation using the formula for message packet setup and delivery in [Dun86][1]. The interstage communication time is taken to be the greatest communication time reported by any node within a stage. The sum of mean image production time for each stage and the greatest communication time for the stage during pipeline operation yields the total time spent in the stage. The sum of these times for each stage results in the total elapsed time to render an image.

There are two components to the performance of the image processing pipeline I propose: The image production rate and the elapsed time through the pipeline. The image production rate, T_r, is the rate at which images emerge from the pipeline and is equal to the processing speed of the slowest stage in the pipeline. The elapsed time through the pipeline, T_e, is defined as the amount of time which must elapse before the image corresponding to a user input is displayed, ie., the time required for an input to be converted into an image. Both components are used to describe pipeline performance when in a steady state, ie., when producing a continuous stream of images and not when loading image data.

5.3 The Mesh and Hypercube Interconnection Topologies.

The image processing pipeline architecture described in the proceeding sections is implemented on a HP-9000 Model 300 running HPUX™ 6.21. All of the code is written in C™, with communication between stages accomplished using inter-process message passing to simulate the transmittal of message packets.

The mesh and hypercube parallel processing node interconnection topologies were evaluated for their suitability in a medical image processing environment. These topologies

[1]The time required to transmit N bytes between two adjacent nodes is 446.7 + 2.4N microseconds.

were selected for evaluation because they are *regular* interconnection topologies and have been implemented in commercially available machines. Each topology is assumed to be embedded in a hypercube- or mesh-interconnected multiprocessor machine. As each stage of the pipeline described in Section 5.2 uses different software, the hypercube and mesh multiprocessor machines are required to be MIMD multiprocessors. Because of system limitations in the HP software, the full-scale medical imaging pipeline described in the preceding section could not be implemented. Instead, a subset of the pipeline containing all the required stages was formulated and this abbreviated pipeline is depicted in Figure 5.3.

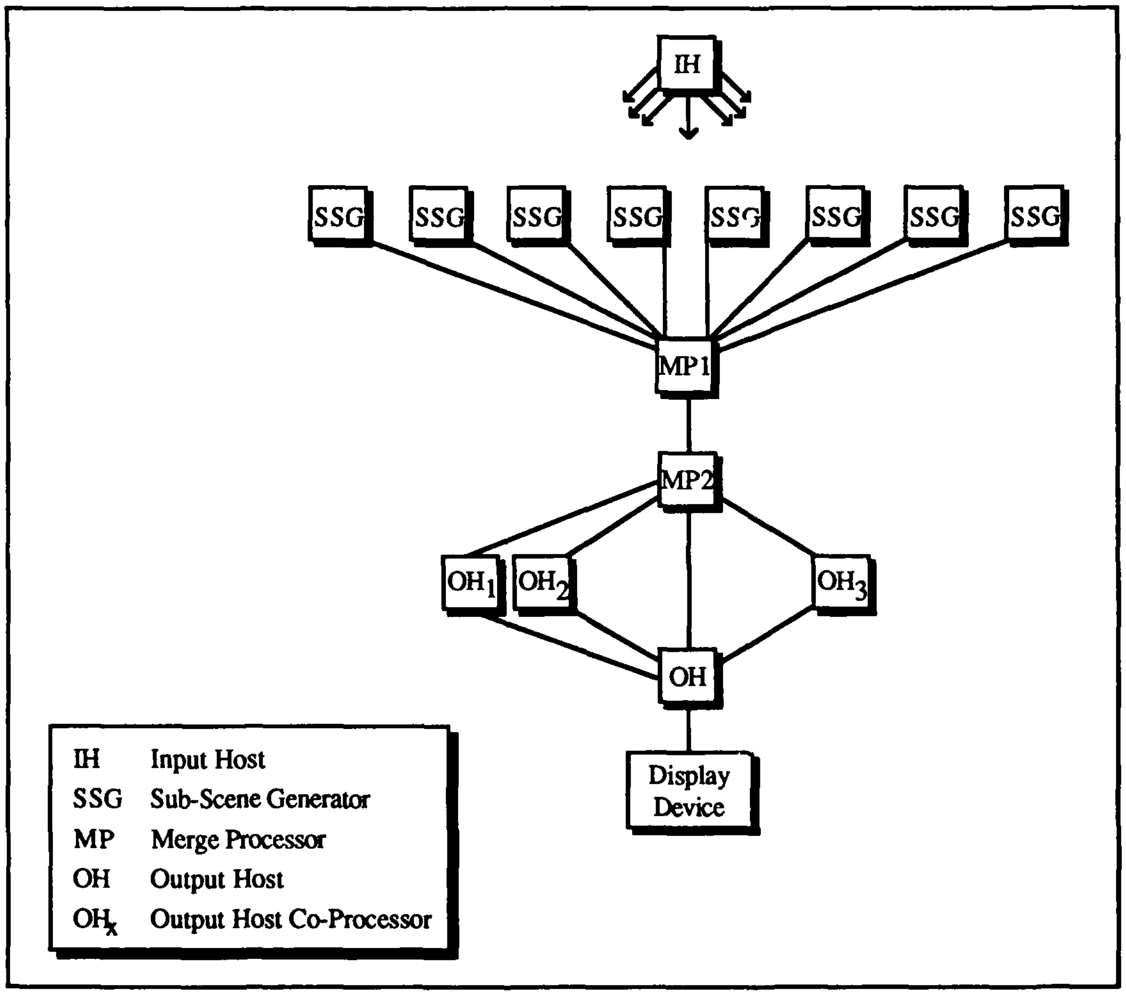


Figure 5.3: Abbreviated Image Processing Pipeline.

However, this simplification of the pipeline yields meaningful results because medical image data permits a data-parallel approach, as defined in [Tuc88], for programming the machine. This approach allows the PEs in each stage to operate in concert without interference from other pipeline stages. Therefore, the timing results for a given stage remain valid as the overall image size increases because the amount of work performed by a PE in a given stage remains constant. Elapsed times for later stages of the full-scale pipeline can be gathered as described in Section 5.2, even though they do not appear in the abbreviated pipeline.

To accommodate the abbreviated pipeline, 16-node mesh and hypercube machines are assumed. The interconnection topologies for these machines are depicted in Figures 5.4 and 5.5.

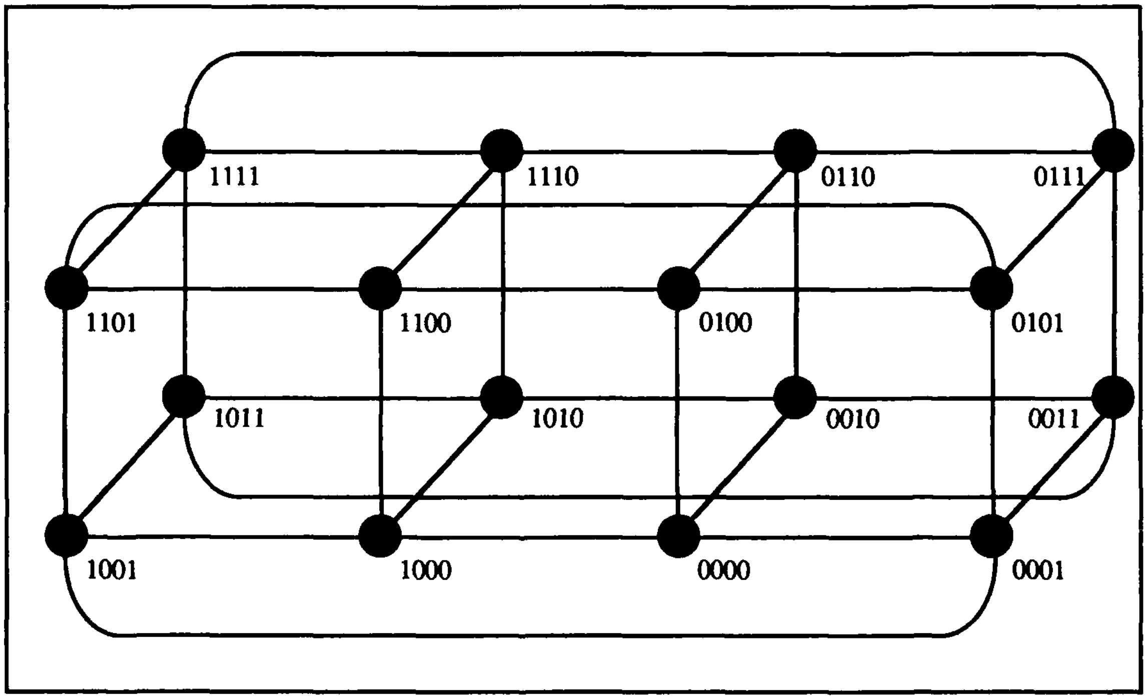


Figure 5.4: Hypercube Interconnection Topology.

Four assumptions are made concerning the environment provided by the multiprocessor machines. First, I assume that the amount of memory at each node is

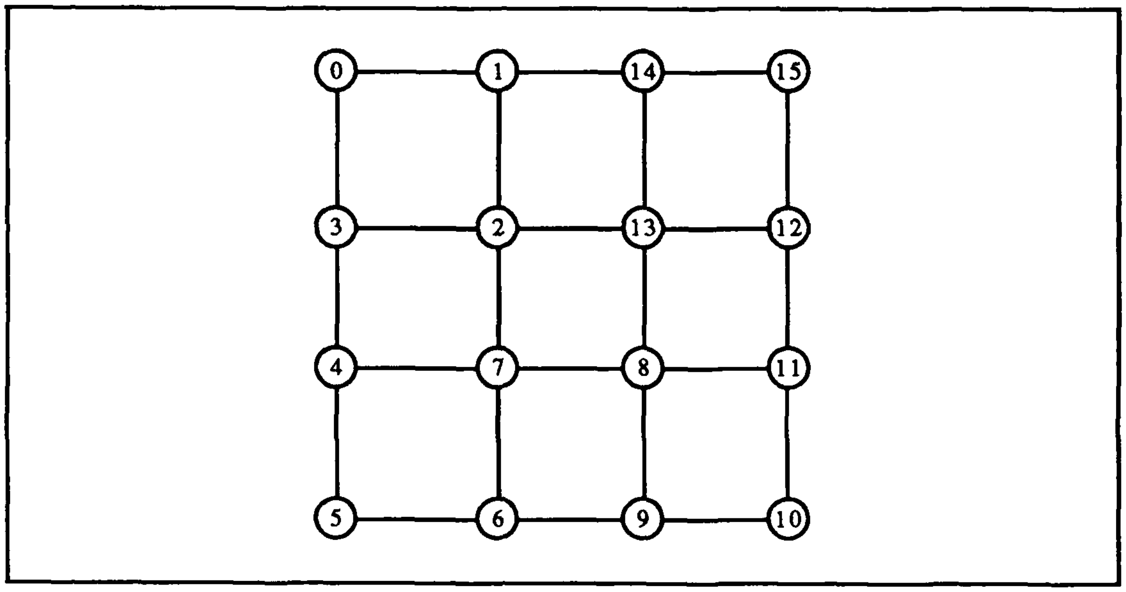


Figure 5.5: Mesh Interconnection Topology.

sufficient to meet the storage requirements of the algorithms in each stage. This is a valid assumption for three reasons. First, the cost of memory continues to decrease. This decreasing cost makes it practical to place several megabytes of memory at each node of a multinode machine. Second, the advent of 16 Mbyte chips makes dedicating large amounts of memory to each processor feasible from the viewpoint of the space and power requirements. For example, the CalTech Mark III Hypercube currently assigns well over 4 megabytes of memory to each node and the recently announced NCUBE 2 series provides 1 - 16 Mbytes/node [Ncu89]. Finally, the current generation of microprocessors is capable of using this large address space directly. Therefore, the memory requirements which the volume rendering algorithms impose can be met by currently available technology.

Second, each mesh and hypercube node is assumed to have a dedicated disk available. This assumption is valid because there are machines like the InTel IPSC/2 currently on the market which possess this capability.

The third assumption is that the machine has at least six internode communication links, or equivalently is a 64-node machine, whenever the full-scale pipeline is discussed. This assumption is valid because machines with many more nodes are in existence. This assumption is made because it simplifies mapping the full-scale pipeline into either architecture.

Fourth, each node of the mesh and hypercube machines is assumed to have at least the processing power of the CPU running the simulation. This assumption is supported by the existence of more powerful processors in currently operating machines. For example, each processor in an NCUBE 2 series is rated as 7 MIPS machine [Ncu89], and the 68020 on the HP 9000 is estimated to be a 1 MIP machine.

The pipeline elements are mapped to the mesh and hypercube architectures so that the communication cost between stages is minimized. Table 5.1 contains the pipeline-node-to-machine-node mapping for both interconnection topologies.

Table 5.1
Image Processing Pipeline to Multiprocessor Architecture Mappings

Pipeline Node	Hypercube Node(s)	Mesh Node(s)
Input Host	0	8
Sub-Scene Generators	1,2,3,4,5,6,8,9	0,1,2,3,4,5,6,9
Merge Processor 1	7	7
Merge Processor 2	15	12
Output Host	12	13
Output Host Co-Processor	11,13,14	10,11,14

5.4 Initial Performance Results.

To determine the nominal interconnection topology and to identify processing bottlenecks in the pipeline, the abbreviated pipeline was embedded within each interconnection topology employing a wide range of packet sizes for inter-processor communication. The conclusions presented in this section are based upon message packet

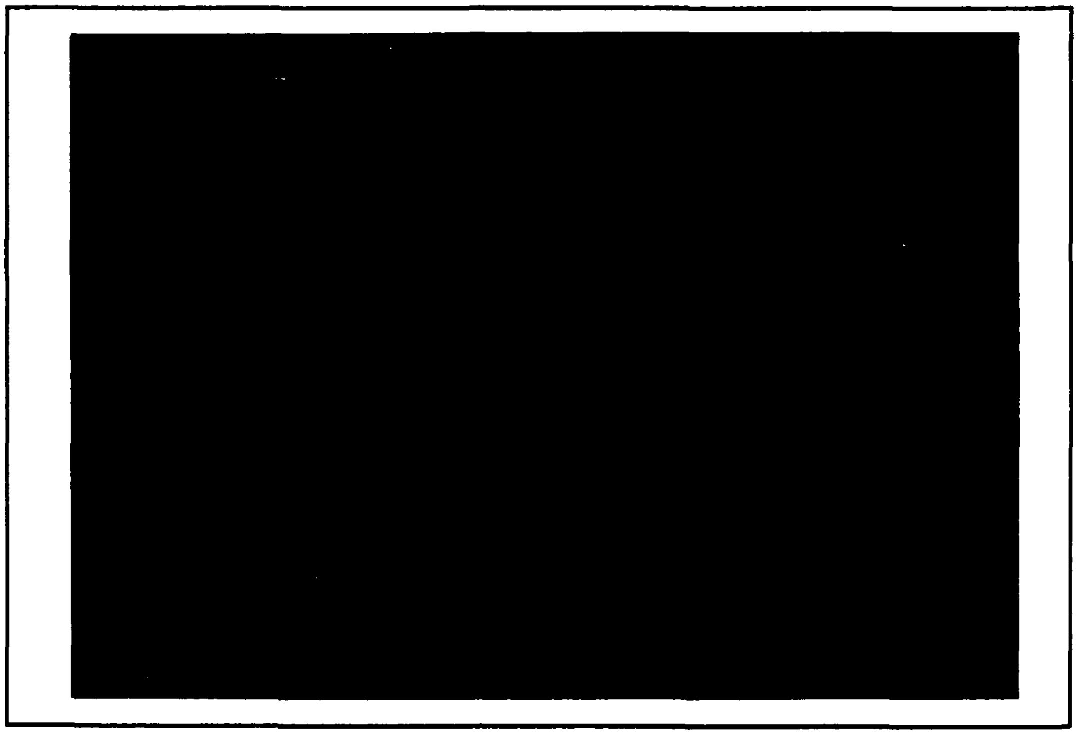


Figure 5.6: Human Vertebrate in Gel Rendered in the Hypercube Interconnection Topology Using the Basic Back-to-Front (BTF) Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 0° z-axis Rotation.

sizes of 512 bytes, 4k bytes, 16k bytes, and 64k bytes with a 16^3 voxel scene at each SSG, for a total scene volume of 32^3 voxels[1] processed within the pipeline. SSG performance was improved over that obtained by a naive implementation of the pseudocode algorithm in Appendix C by computing the shifted SCT values for each recursion level and storing these results in a table indexed by recursion level and octant number. As hidden-surface removal is known to be computationally intensive, two different variations of each of the basic HSR algorithms were investigated. These four implementations are the Back-to-Front (BTF) algorithm using a subcube center (SCC) octant generator, the BTF algorithm using a front-lower left (FLL) octant generator, the Front-to-Back (FTB) algorithm using a SCC octant generator, and the FTB algorithm using a FLL octant

[1]In the remainder of this work, scene and volume sizes are assumed to be given in voxels, exceptions are explicitly noted.

generator. Using the basic recursive BTF SCC HSR algorithm, a 64^3 voxel volume depicting a human vertebrate in gel was rendered using the hypercube interconnection topology with 64k byte message packets at 0° rotation. The resulting image is presented in Figure 5.6.

Using the basic recursive BTF SCC HSR algorithm, a 64^3 voxel volume depicting a mitre box was rendered using the hypercube interconnection topology with 64k byte message packets at 0° rotation as well as 45° x-axis and 45° y-axis rotation. The resulting images are in Figures 5.7 and 5.8.

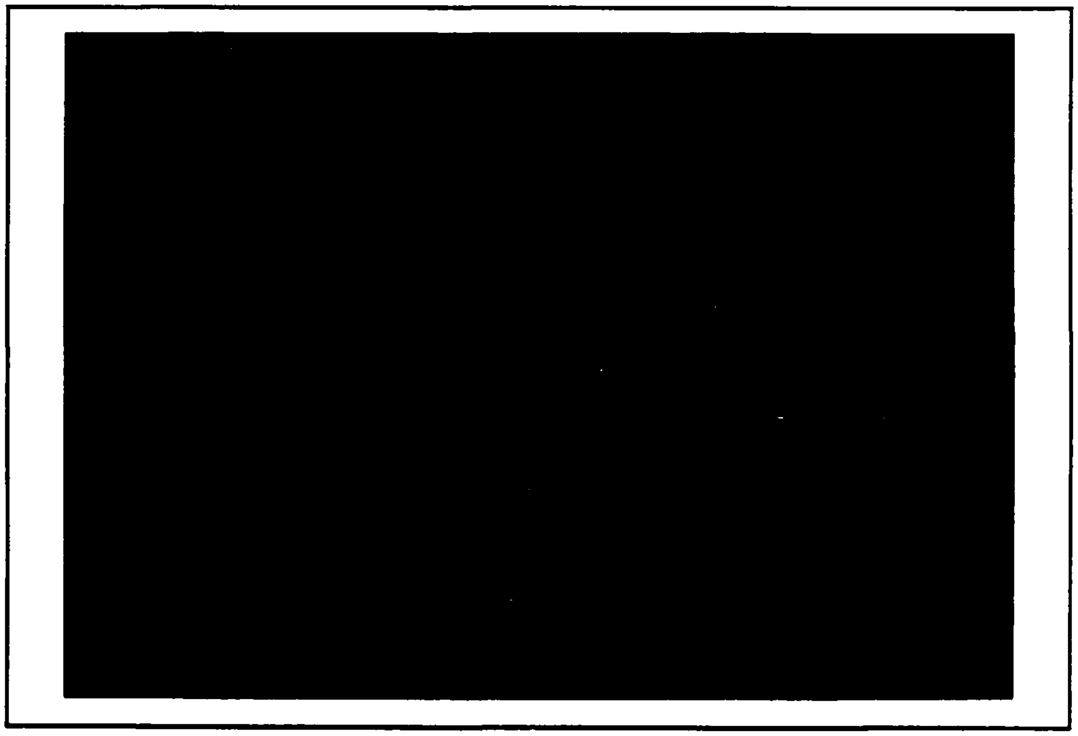


Figure 5.7: Mitre Box Rendered in the Hypercube Interconnection Topology Using the Basic Back-to-Front (BTF) Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 0° z-axis Rotation.

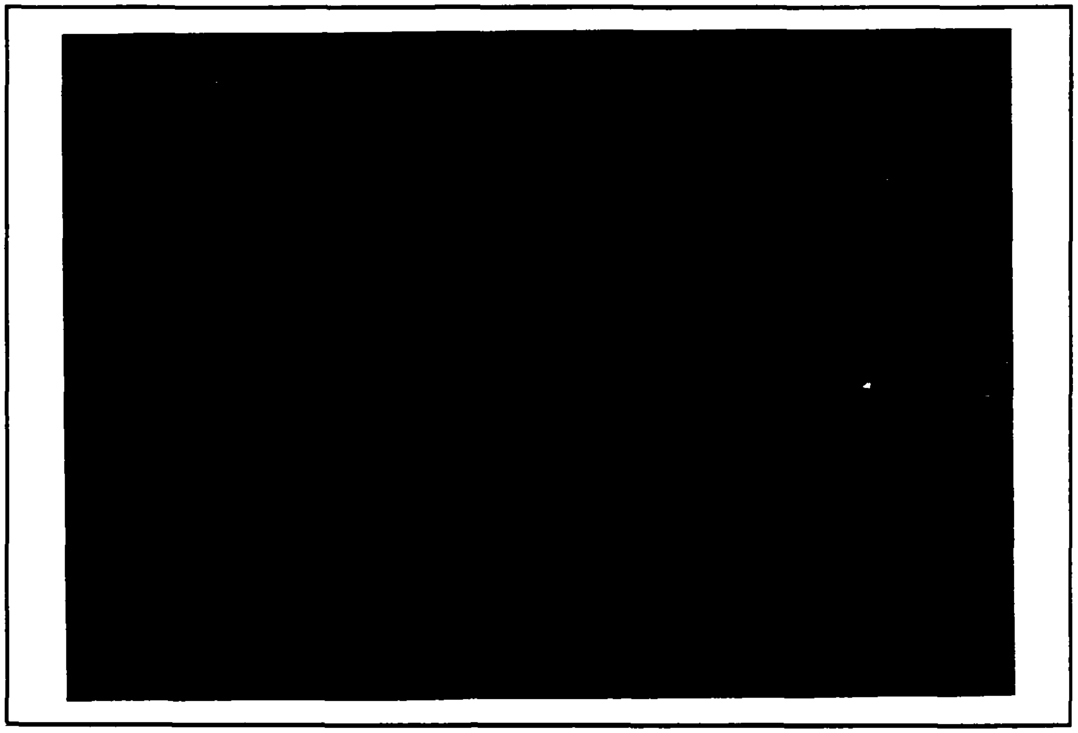


Figure 5.8: Mitre Box Rendered in the Hypercube Interconnection Topology Using the Basic Back-to-Front (BTF) Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 45° x-axis and 45° y-axis Rotation.

Communication and computation times for each stage are obtained using the methodology presented in the section 5.2.4. The complete set of timing results for the four different implementations are contained in Appendix D and demonstrate that message packet size has negligible effect on pipeline performance. Further discussion of pipeline performance and interconnection topology preference can, therefore, be based upon a single packet size. The 64k packet size was selected because these results generalize to larger image sizes with little increase in total number of packets generated. The following discussion is based upon the 64k packet size results for pipeline throughput and individual stage performance presented in Figures 5.9 and 5.10.

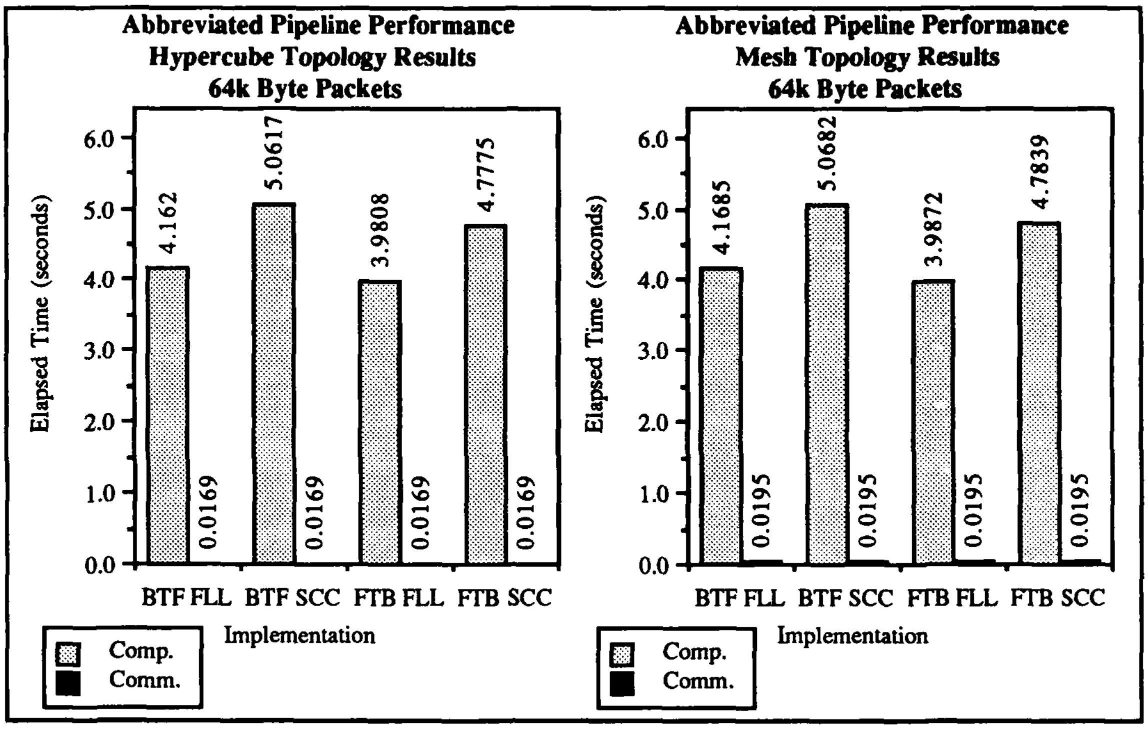
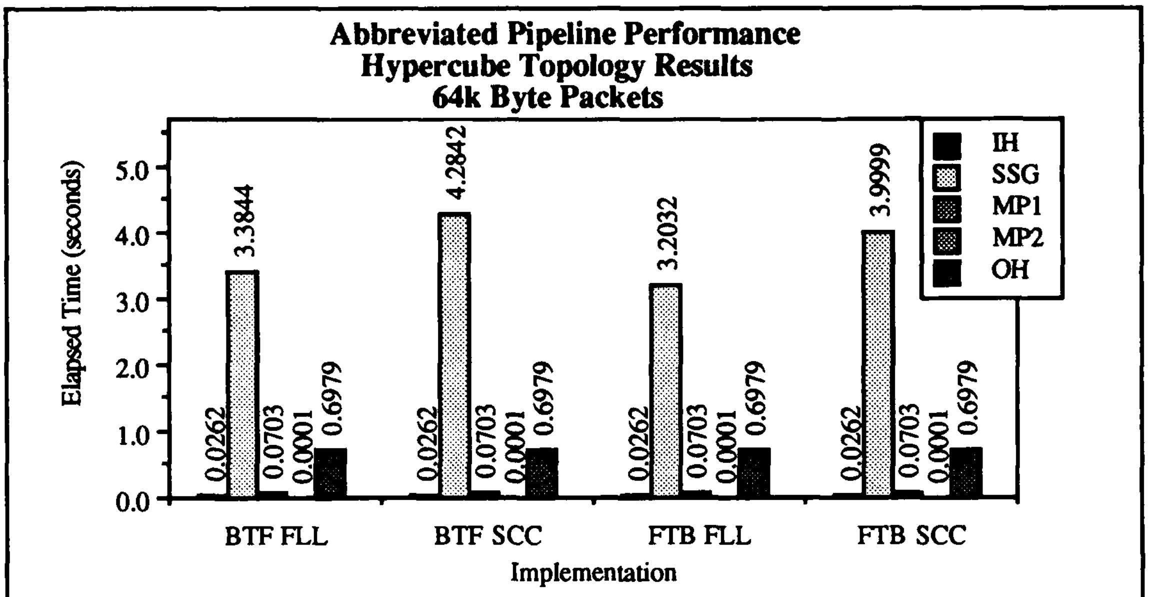


Figure 5.9: Image Processing Pipeline Throughput Performance Within the Hypercube and Mesh Interconnection Topologies.

These results demonstrate that communication costs, and hence multiprocessor interconnection topology, have an insignificant impact on image rendering time and that the SSG stage is the performance bottleneck in the image processing pipeline. As the SSG stage has the largest stage processing time, T_r for all four implementations is the SSG stage time. Since approximately 80% of the elapsed image rendering time is spent in the SSG stage, and because OH operation can be sped up with hardware shading, T_e for the pipeline can be usefully approximated by SSG stage time. To determine how SGG performance scales to different image sizes, SSG performance on 16^3, 32^3 and 64^3 volumes[1] for each of the four algorithms was examined. These results are contained in Table 5.2, and summarized graphically in Figure 5.11.

[1]These volume sizes correspond to 8^3, 16^3 and 32^3 voxel data volumes before supersampling.



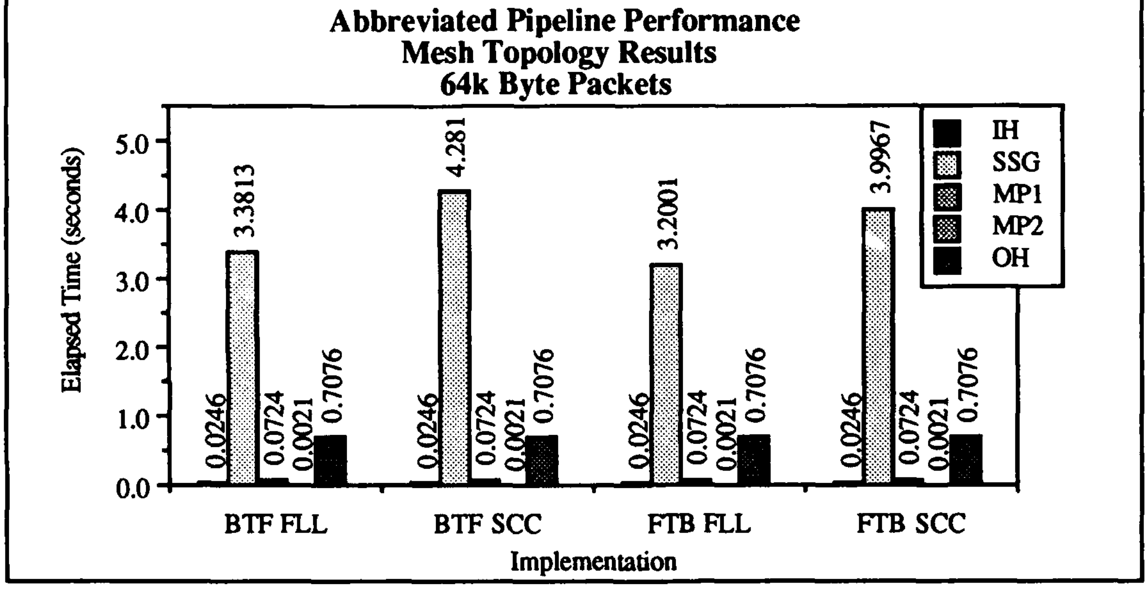


Figure 5.10: Image Processing Pipeline Stage Performance Within the Hypercube and Mesh Interconnection Topologies.

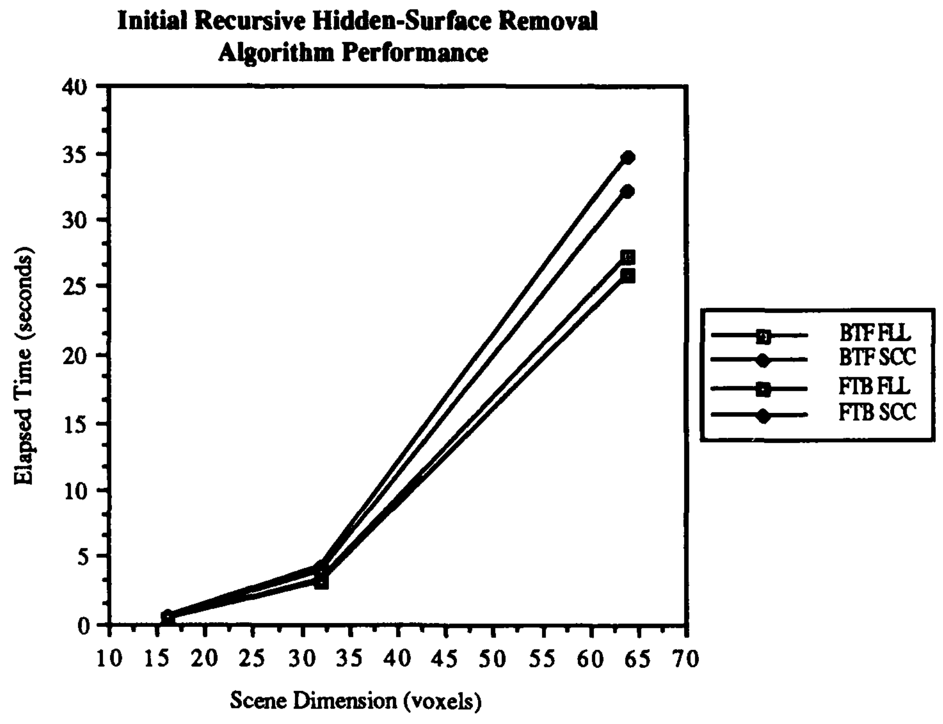
Note that the amount of time spent at the SSG stage varies linearly with the change in scene size at the SSG and that the FLL implementation of the BTF and FTB algorithms operates approximately 20% faster than the corresponding SCC implementation. This difference occurs because the FLL approach requires fewer computations at the deepest

Table 5.2
Elapsed Image Rendering Time Performance of the Basic Recursive Hidden-Surface Removal Algorithms (in seconds).

Supersampled SSG Scene Size	ALGORITHM			
	Back-to-Front, FLL	Back-to-Front, SCC	Front-to-Back, FLL	Front-to-Back, SCC
16^3	.420 seconds	.53 seconds	.40 seconds	.50 seconds
32^3	3.363 seconds	4.26 seconds	3.18 seconds	3.98 seconds
64^3	27.31 seconds	34.75 seconds	25.81 seconds	32.15 seconds

level of recursion than the SCC approach. However, these results demonstrate that the performance of the SSG stage using the basic HSR algorithm effectively eliminates the possibility of near-real time medical image generation on a commercial multiprocessor.

At this point, I reassessed the decision to employ recursive HSR algorithms in the SSG stage with a view toward determining whether to abandon this approach or try to modify the algorithms. Because recursive HSR algorithms are partitionable and processor independent, which are desirable properties in a MIMD multiprocessor environment, I decided to investigate improving the performance of the recursive HSR algorithms. To gain a deeper insight into the operation of the class of recursive HSR algorithms, I examined the published literature on these algorithms with the goal of locating definitions and research which provide insight into their performance. Finding none, and requiring a guide to assist me in my development and presentation of improved recursive HSR algorithms, I fashioned my own definition of the class of recursive hidden-surface removal algorithms. This definition is presented in the following section.



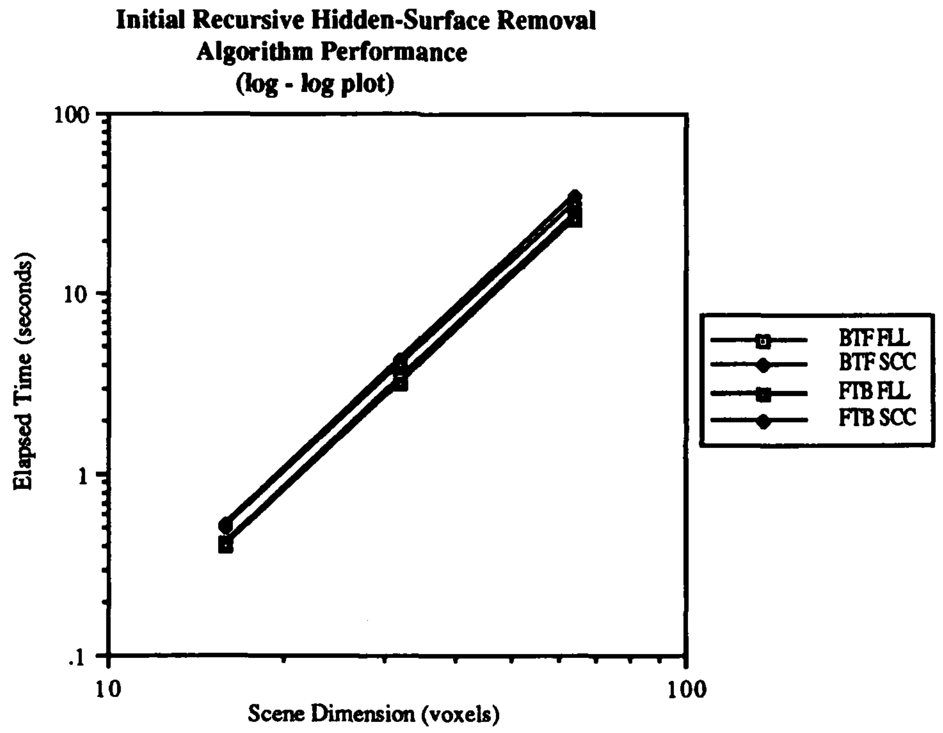


Figure 5.11: Performance of the Basic Recursive Hidden-Surface Removal Algorithms.

5.5 Recursive Hidden-Surface Removal Algorithms Defined.

This section opens with examples which demonstrate the operation of recursive hidden-surface removal algorithms. This presentation is followed by a general definition of recursive HSR algorithms, using the concepts of basis vectors and sets of points. The section concludes with some short notes on implementation differences between the front-lower left (FLL) octant generator location and sub-cube center (SCC) octant generator location variations.

The operation of recursive hidden-surface removal algorithms is dependent upon the use of recursive coordinate composition. Figure 5.12 illustrates the concept of recursive coordinate composition in two-space upon a scene which is a power of two in each dimension. For this example, the objective is to determine the coordinate value of the shaded point. The coordinate computation begins using the two components, V_x and V_y, of the vector V, which is a vector directed from the origin to the front lower left of the upper right-hand (third) quadrant. Let $Q_{i,j}$ be the quadrant j within quadrant i, and Q_{c_j} be the indicated coordinate for the object space front lower left voxel of quadrant j. Initially, V_x and V_y point to the front lower left of quadrant 3, which lies at coordinates Q_{x_3}, Q_{y_3}. The next step in determining the coordinates of the shaded point requires halving V_x and V_y, yielding $V_x/2$ and $V_y/2$. By adding $V_x/2$ and $V_y/2$ to Q_{x_3}, Q_{y_3}, or alternatively to V_x and V_y, the coordinate value for the voxel at the front lower left of quadrant 3 within the current quadrant , which lies at coordinates $Q_{x_{3,3}}$, $Q_{y_{3,3}}$, is obtained. The next step toward establishing the desired coordinates of $Q_{x_{3,3,3,3}}$, $Q_{y_{3,3,3,3}}$ requires computing $Q_{x_{3,3,3}}$, $Q_{y_{3,3,3}}$ by calculating $V_x/4$ and $V_y/4$ and adding this result to $Q_{x_{3,3}}$, $Q_{y_{3,3}}$. A fourth, and final, application of the power-of-two-scaling/summing operation to both V_x and V_y yields the desired coordinate values for $Q_{x_{3,3,3,3}}$, $Q_{y_{3,3,3,3}}$. The other three points in quadrant $Q_{3,3,3}$ are computed as follows. $Q_{x_{3,3,3,2}}$, $Q_{y_{3,3,3,2}}$ is computed by adding $V_y/8$ to

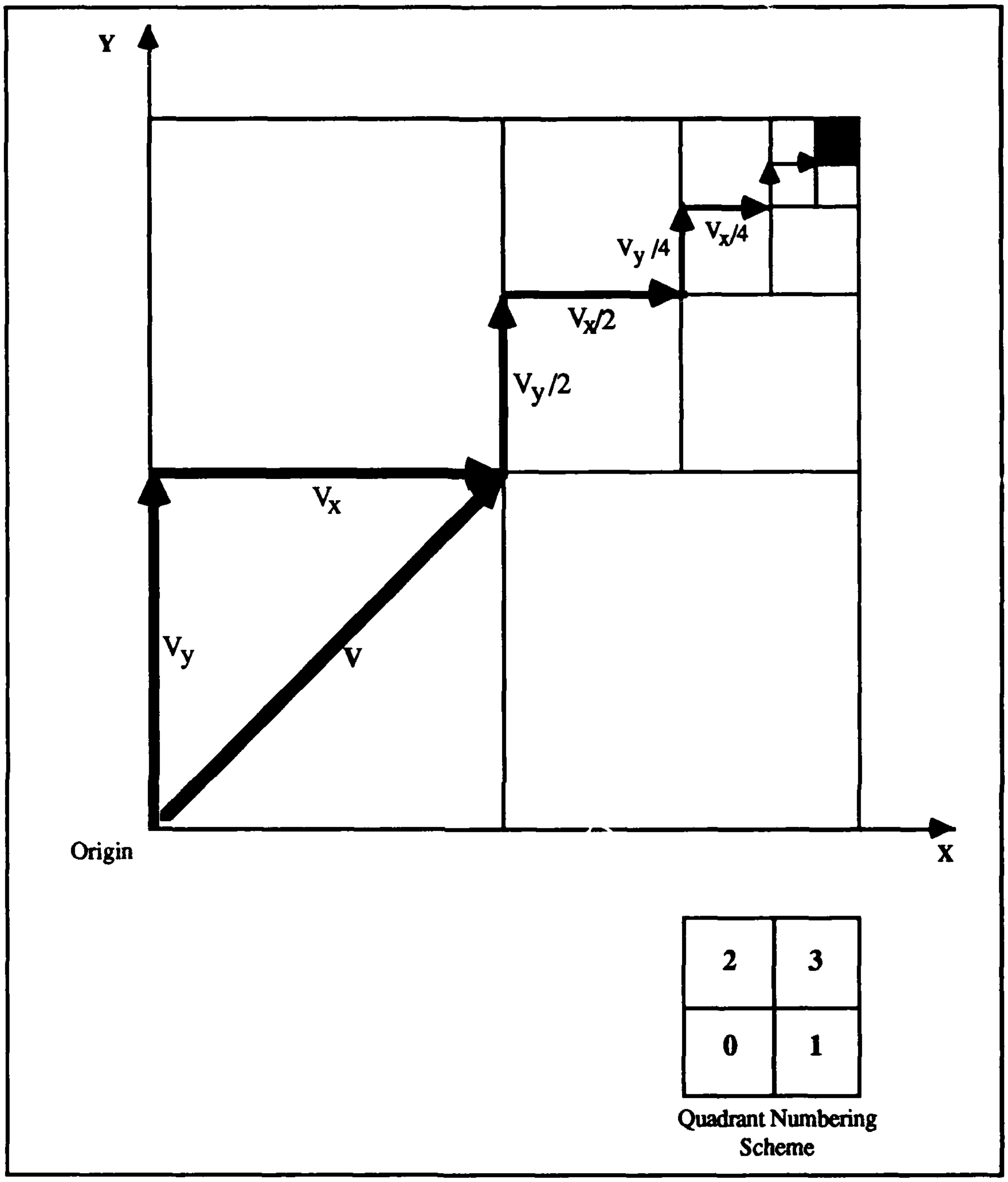


Figure 5.12: Determining a Coordinate by Composition.

$Q_{y_{3,3,3}}$ and leaving the $Q_{x_{3,3,3}}$ coordinate unchanged. $Q_{x_{3,3,3,1}}$, $Q_{y_{3,3,3,1}}$ is computed by adding $V_x/8$ to $Q_{x_{3,3,3}}$ and leaving the $Q_{y_{3,3,3}}$ coordinate unchanged. $Q_{x_{3,3,3,0}}$, $Q_{y_{3,3,3,0}}$ is computed by leaving both the $Q_{x_{3,3,3}}$ and the $Q_{y_{3,3,3}}$ coordinates unchanged. By interchanging the summing operation with a null operation for both, or either, of the vectors at each point in the computation, every point in the space can be plotted so long as the initial vector set is a basis vector set for the space. Note also that the number of recursive coordinate compositions required (4) is equal to $\log_2$(scene dimension).

The other component of the recursive hidden-surface removal strategy is the use of back-to-front, or front-to-back, image space processing to insure that only the voxels visible from the desired orientation are contained in the rendered image. The back-to-front approach requires that image space processing begin at the rearmost voxel in image space, and that successive voxel values are written to screen space in the image space back-to-front order. The front-to-back technique, on the other hand, begins processing at the foremost image space voxel and allows voxel values to be written to a given screen space location only if no voxel value has been previously written to the location. For both techniques, the rendered image contains only the density values of the voxels visible from the given orientation.

For both the back-to-front and front-to-back algorithms, a critical decision is the selection of technique for computing the foremost (rearmost) image space voxel and then moving foreword (rearward) in image space in an ordered manner. A recursive back-to-front (front-to-back) hidden-surface removal algorithm uses recursive coordinate composition, as described above, to locate the rearmost (foremost) voxel and impose the required processing sequence in image space. Having made this choice, the same technique is also used in object space so that movement through both spaces can proceed in parallel. As the object space coordinates are generated, they are mapped into a 3D array wherein the voxel values are stored. These voxel values are then written into the 2D screen space array at the u, v coordinates given by the x', y' image space coordinates. The z' image space coordinate value is retained for shading and distance comparison in front-to-back processing. Note that in order for this technique to work correctly, the object space vector corresponding to each image space vector must be known.

The decision to use recursive coordinate composition imposes the twin restrictions that object space and image space be cubical and a power of two in dimension and that a basis set of vectors for both spaces be available. The first restriction can be met by a combination of modality slice interpolation and/or padding of the 3D object space array with data from void slices. The second restriction is satisfied by insuring that the set of vectors used is a superset of the orthogonal vectors for each space. Two common choices for the vector set are: 1) the set of vectors defined by the lines connecting the center of the scene and the voxel at the center of each octant, and 2) the set of vectors defined by the lines connecting the scene origin (at the front, lower, left of the volume) and the front, lower, left voxel of each octant. To further clarify these points before the definition is presented, the following example conceptually illustrates back-to-front recursive hidden-surface removal processing in 2D.

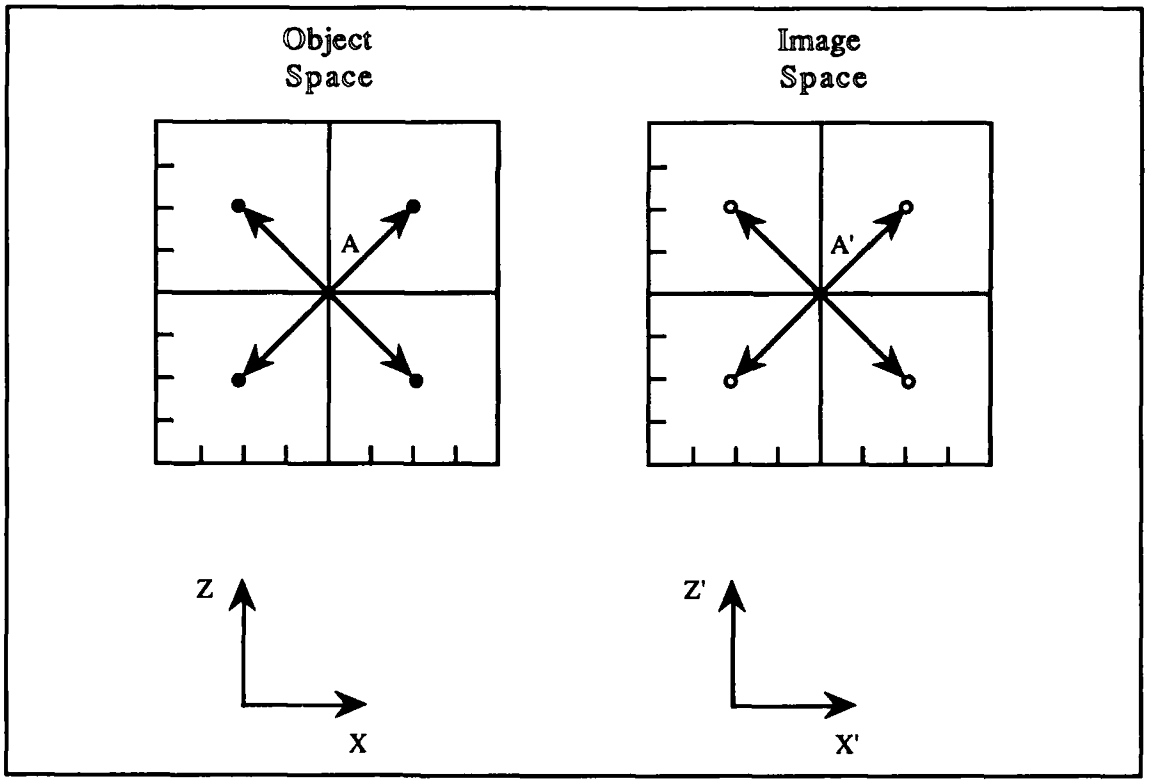


Figure 5.13: Initial State for Subcube-Center-Based Back-to-Front Recursive Hidden-Surface Removal Processing.

Figure 5.13 contains a 2D representation of object and image space, with each space being 8 voxels in both dimensions. Therefore, the required recursion depth is three.

The basis coordinate composition vectors in both spaces are indicated by arrows which run from the scene center to the center of each scene quadrant. For ease of exposition and clarity, the example assumes an unrotated scene, so the image space and object space vectors are identical. In order to compute the coordinates of the rearmost voxel in image space, processing requires selection of the vector with the deepest z' penetration in image space at each step in the rearmost voxel computation processing. Whenever image space vectors have equal z' penetration, the ties are broken arbitrarily, with the restriction that the back-to-front ordering thereby imposed on the vector set is maintained for the remainder of the rendering operation. Because the image space vectors in quadrants 2 and 3 have equal z' penetration, an arbitrary selection of deepest vector is made. The tie is broken by selecting the vector labeled A' in image space, forcing the selection of the vector labeled A in object space as the initial object space vector. The back-to-front order for the remaining vectors is established, for this example, counterclockwise from A'. The choice of the vector A' as the initial vector and the processing on the vectors A' and A completes initial processing at the first level of the recursion. The next processing step requires computation of the center coordinate of the rearmost subquadrant lying within the quadrants containing A' and A.

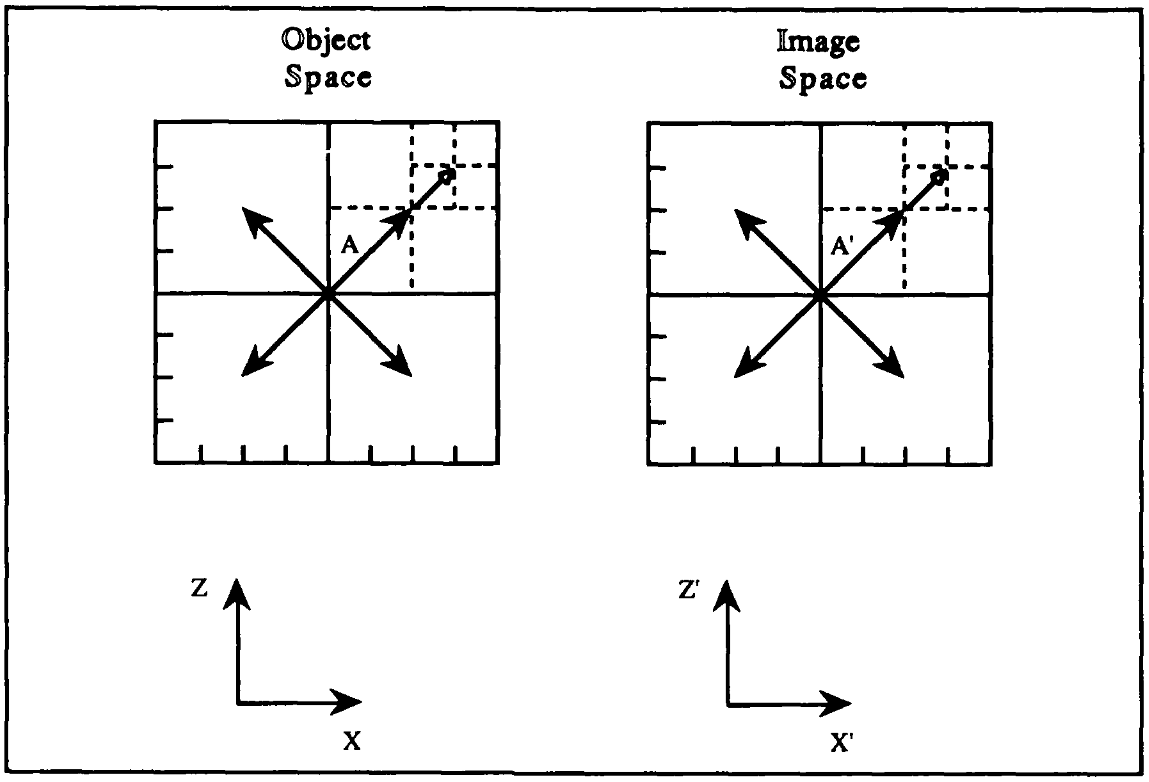


Figure 5.14: Subcube-Center-Based Back-to-Front Recursive Hidden-Surface Removal Processing at the Second Level of Recursion.

Figure 5.14 depicts the processing status in both spaces at the conclusion of the first pass within the second level of recursion. For clarity, the subquadrants surrounding the center coordinate of each quadrant are indicated by dashed lines. The center coordinate value for the rearmost subquadrant in image space is determined by scaling the deepest z' vector by $1/(2^{\text{depth of recursion}})$, and adding the resulting vector values to the initial vector value at the current recursion depth. Therefore, since the current recursion depth is two, the components of the vector A' are scaled by 1/4 and then added to the coordinate values lying at the endpoint of A'. The same halving/summing operation is performed concurrently in object space using the vector A.

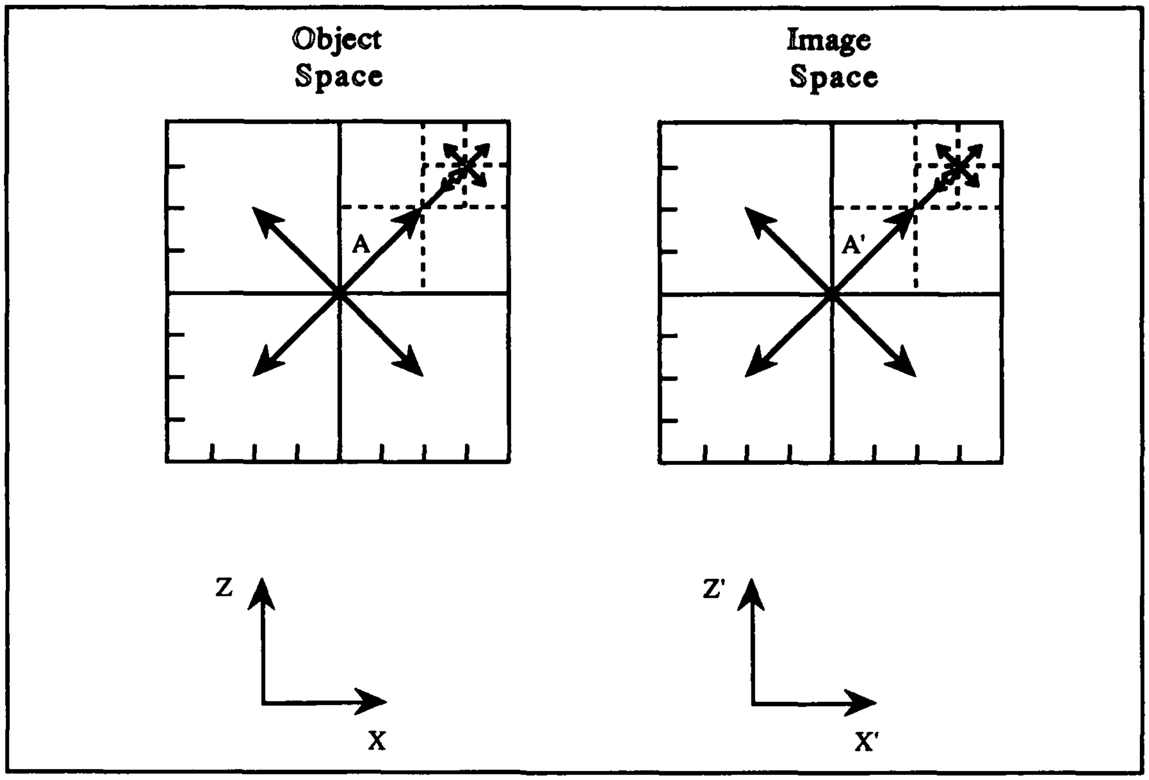


Figure 5.15: Subcube-Center-Based Back-to-Front Recursive Hidden-Surface Removal Processing at the Third Level of Recursion.

Figure 5.15 depicts the image rendering state at the completion of processing of the deepest, third, level of recursion. At the deepest level of recursion, the processing operation changes. Because this quadrant will never be reached again during the remainder of the image rendering operation, all the subquadrants within the level two recursion quadrant must be processed before returning to the next higher level of the recursion. This processing is accomplished by selecting each of the image space vectors in turn according to their back-to-front order, scaling them appropriately, and adding the scaled result to the quadrant center coordinate value, thereby computing the value of the center coordinates of each terminal subquadrant. In Figure 5.15, the coordinates are computed counterclockwise from the quadrant in the furthest upper right-hand corner of image space. As each image space/object space coordinate set is computed, the voxel density value stored within the object space array, at the position given by the object space coordinate values, is mapped into screen space according to the image space coordinates. Once all four subquadrants are processed, the recursion returns to the next higher level, two, to process the next area.

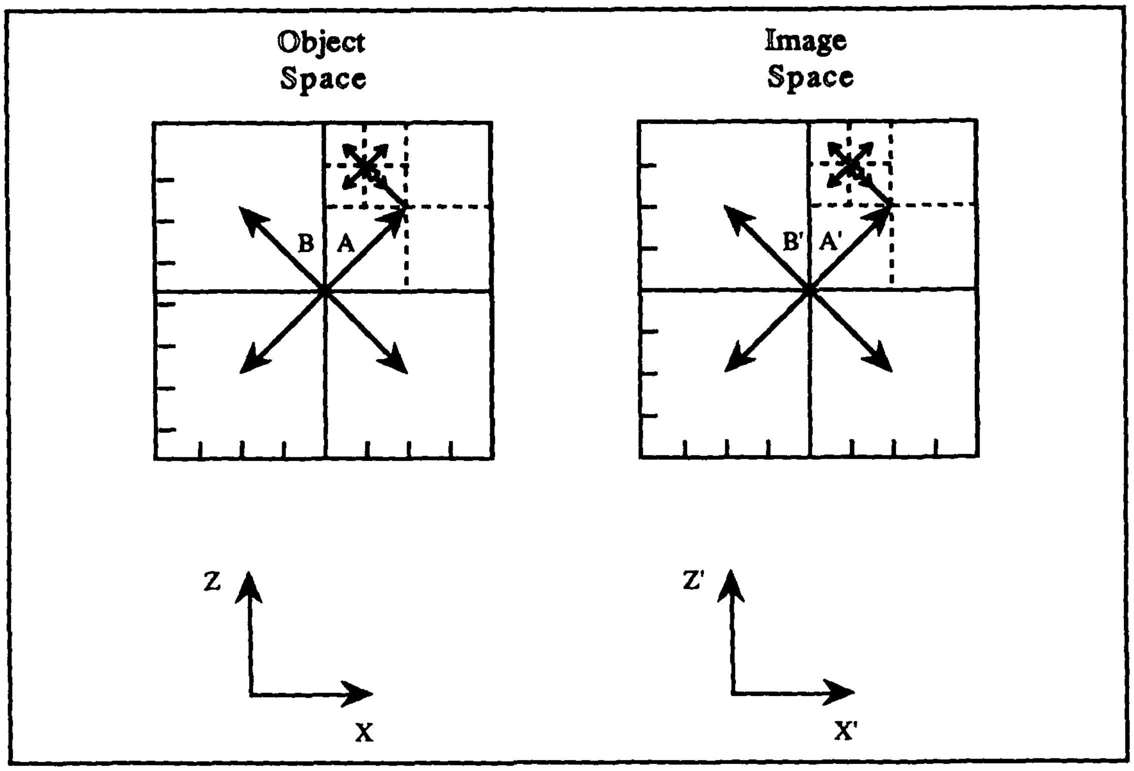


Figure 5.16: Subcube-Center-Based Back-to-Front Recursive Hidden-Surface Removal Processing at the Third Level of Recursion: Second Pass.

Since all four subquadrants defined at the current level, two, of the recursion have not been processed, the next deepest subquadrant at level two is selected for processing. Because processing must proceed in back-to-front order, the next vector selected for scaling is the vector with the second deepest z' penetration. This vector was previously defined to be the vector counterclockwise from the vector A'. Therefore, the vector labeled B' is scaled by 1/4 and added to the center coordinate value of the current quadrant, yielding the center coordinate value for the next deepest subquadrant within the current quadrant. This computation achieves the deepest level of recursion, so the center coordinate values of all four sub-subquadrants are computed and the object space to screen space mapping is accomplished. These processing steps are illustrated in Figure 5.16.

The class of recursive back-to-front algorithms is defined using partitions of object and image space and a set of operations on these spaces. To hold the initial data set and the resulting image, two arrays are employed, a 3D object space array and a 2D image space array, each of dimension such that for each point in object space and image space there is an array element. The object space partitions are defined using a set of neighborhoods of points in object space, $\{\mathcal{N}_O\}$, such that each neighborhood is a regular shape and the set of neighborhoods completely tessellates object space. One of the simplest shapes with which to tessellate a 3D space is a cube, which both I and [Fri85a] employ to define the neighborhoods. The use of a cube partitions object (and image) space into octants. The components of the definition are illustrated in 2D space in Figure 5.17.

To generate the set of points in each $\{\mathcal{N}_O\}$, a set of basis vectors and a starting location for the generation of the set are required for each neighborhood. This requirement is met for each $\{\mathcal{N}_O\}$ by selecting a point $P_{0,}$ and defining the vector V_O as the vector aligned from the origin to the point P_0. This selection process generates the set $\{\mathcal{P}_O\}$ of points and the set $\{\mathcal{V}_O\}$ of vectors in object space. While $\{\mathcal{V}_O\}$ is not a basis set, the choice of a cube as the basic tessellation element guarantees that $\{\mathcal{V}_O\}$ contains a basis set. To generate all the points in object space, determine a relation $\mathcal{R}$ such that all the points in $\{\mathcal{N}_O\}$ can be computed by applying $\mathcal{R}$ to $\{\mathcal{P}_O\}$ and $\{\mathcal{V}_O\}$. The vector set and generation points required for image space are computed by applying the rotation matrixes, M, to each of the points in $\{\mathcal{P}_O\}$, ie. P_0 x M -> P_I, resulting in the image space coordinates for the set of points in $\{\mathcal{P}_O\}$. This new set of points $\{\mathcal{P}_I\}$, and its associated vector set $\{\mathcal{V}_I\}$, are used to generate the set of points $\{\mathcal{N}_I\}$ which result when $\{\mathcal{N}_O\}$ is rotated into image space. Determine the BTF $\{\mathcal{N}_I\}$ generation sequence for $\{\mathcal{P}_I\}$ by sorting the subset of points in

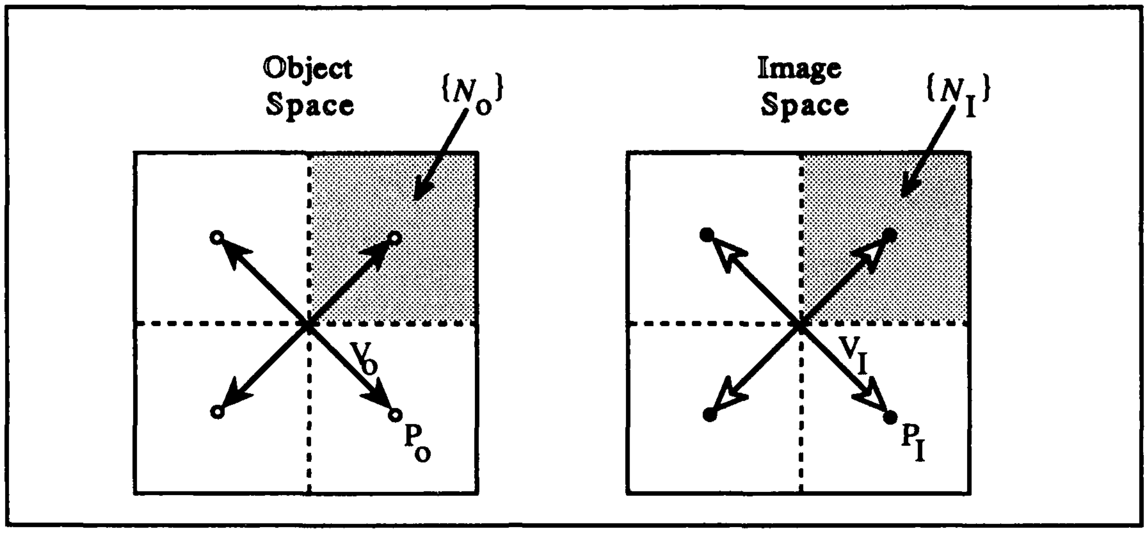


Figure 5.17: Major Components in the Definition of the Class of Recursive Hidden-Surface Removal Algorithms.

the scene contained in $\{\mathcal{P}_I\}$ in order of decreasing distance from the observer. Starting with the point in $\{\mathcal{P}_I\}$ furthest from the observer, and with its corresponding point in $\{\mathcal{P}_O\}$, apply the relation $\mathcal{R}$ to $\{\mathcal{P}_I\}$ and $\{\mathcal{P}_O\}$ in parallel, thereby computing $\{\mathcal{N}_O\}$ as well as the image space coordinates $\{\mathcal{N}_I\}$. Whenever $\mathcal{R}$ yields a single point in $\{\mathcal{N}_I\}$ and $\{\mathcal{N}_O\}$, project the data element from the identified position in the object space array to its new location in the image space array. When $\mathcal{R}$ has been applied to all points in $\{\mathcal{P}_O\}$ and $\{\mathcal{P}_I\}$, thereby rotating the whole of object space into image space, the contents of the image space array contain the desired view of object space with all hidden-surfaces removed.

The class of recursive front-to-back algorithms is defined using the same object and image space partitions and operations as employed to define recursive back-to-front algorithms. There are two noteworthy differences. First, the FTB sequence for processing the points in $\{\mathcal{P}_I\}$ is determined by sorting $\{\mathcal{P}_I\}$ in order of increasing distance from the observer. Second, once $\mathcal{R}$ has produced a single point in $\{\mathcal{N}_I\}$ and $\{\mathcal{N}_O\}$ and it has been projected from the object space array to its new location in the image space array, no other point in $\{\mathcal{N}_O\}$ is allowed to project to that same point in the image space array (screen space). Incorporation of these differences into the definition of the class of recursive BTF algorithms presented above results in a definition for the class of recursive front-to-back algorithms.

There are many choices for $\mathcal{R}$ which may be used to generate the points in $\{\mathcal{N}_I\}$ and $\{\mathcal{N}_O\}$. A simple $\mathcal{R}$ is obtained if $\{\mathcal{P}_I\}$ and $\{\mathcal{P}_O\}$ are composed of the voxels at either the center or front lower left of each neighborhood[1]. The vector set $\{\mathcal{V}_I\}$ then contains the vectors which have a common endpoint at the origin for the volume and with their other endpoint fixed at one of the points in $\{\mathcal{P}_I\}$. $\{\mathcal{V}_O\}$ is defined similarly. Since the vectors are in 3-space, each vector of $\{\mathcal{V}_I\}$ and $\{\mathcal{V}_O\}$ is defined by its three components, V_x, V_y, V_z, with each component lying along its associated major axis. Note that if V_x, V_y, and V_z are scaled by the same amount, then the direction of the vector is preserved and only its length is changed, and that this relationship holds for any scaling factor as long as the same scaling factor is applied to all three components. $\mathcal{R}$ is therefore defined as a recursive scaling operation on $\{\mathcal{V}_I\}$ and $\{\mathcal{V}_O\}$ which forms the vector sets $\{\mathcal{V}_I{}^*\}$ and $\{\mathcal{V}_O{}^*\}$ by halving the length of each of the components of the vectors in the given vector sets using these vectors to compute eight points which lie within eight discrete partitions, $\{\mathcal{N}_O{}^*\}$ and $\{\mathcal{N}_I{}^*\}$, of $\{\mathcal{N}_O\}$and $\{\mathcal{N}_I\}$ respectively[2]. These new points, the $\{\mathcal{V}_I{}^*\}$, and the $\{\mathcal{V}_O{}^*\}$ are passed to the next deeper level of recursion, where eight new partitions of each of the $\{\mathcal{N}_O{}^*\}$ and $\{\mathcal{N}_I{}^*\}$ are generated. The following example illustrates these concepts.

Given the point P in a neighborhood with coordinates C_x, C_y, C_z, a vector set $\{\mathcal{V}\}$, and $\mathcal{R}$, the goal is to compute the coordinates, $C_{x_n}{}^*$, $C_{y_n}{}^*$, and $C_{z_n}{}^*$, of the eight points $P_1 \ldots P_8$ in back-to-front order within the neighborhood such that these eight points

---

[1]The choice of the front lower left cuberille is driven by the choice of the location of the scene origin. A simple R is always obtained if each vector in $\{\mathcal{V}\}$ goes from the origin to the one cuberille closest to the origin in each object space octant as this assignment permits scaling the vectors to shorter lengths without changing their direction or performing additional computations.

[2]Since the original set of $\{\mathcal{N}_O\}$ and $\{\mathcal{N}_I\}$are octants, the $\{\mathcal{N}_O{}^*\}$ and $\{\mathcal{N}_I{}^*\}$ are referred to as suboctants. In general, the $\{\mathcal{N}_O{}^*\}$ at a level ℓ of the recursion are suboctants of a single object space octant at level ℓ-1, with the same relationship applying in image space

have the same spatial relationship to P as P has to the origin. These coordinates can be computed by selecting the elements of $\{\mathcal{V}\}$ in back-to-front order and then using $\{\mathcal{V}\}$ to compute $C_{x_n}{}^* = C_x + V_{x_n}/2$, $C_{y_n}{}^* = C_y + V_{y_n}/2$, and $C_{z_n}{}^* = C_z + V_{z_n}/2$. Note that this operation can be applied recursively. For example, at a given level, ℓ of the recursion, compute eight $C_{x_n}{}^*$, $C_{y_n}{}^*$, and $C_{z_n}{}^*$ for the given P_0 using each element of $\{\mathcal{V}^*\}_\ell$ in BTF order. As each set of $C_{x_n}{}^*$, $C_{y_n}{}^*$, and $C_{z_n}{}^*$ are computed, pass these coordinate values along with $\{\mathcal{V}\}$ to level ℓ+1. At level ℓ+1, compute $\{\mathcal{V}^*\}_{\ell+1}$ by setting $V_x{}^* = V_x/2^{\ell+1}$, $V_y{}^* = V_y/2^{\ell+1}$, and $V_z{}^* = V_z/2^{\ell+1}$ for each element of $\{\mathcal{V}\}$. Using $\{\mathcal{V}^*\}_{\ell+1}$ and the $C_x{}^*$, $C_y{}^*$, and $C_z{}^*$, compute $C_{x_n}{}^{**}$, $C_{y_n}{}^{**}$, and $C_{z_n}{}^{**}$ in the same manner as $C_{x_n}{}^*$, $C_{y_n}{}^*$, and $C_{z_n}{}^*$ were computed. By recursing until ℓ equals $\log_2(N)$, all the points in the given neighborhood can be generated by applying $\mathcal{R}$ to $\{\mathcal{V}\}$ and P.

The $\mathcal{R}$ defined above can be used to accomplish hidden-surface removal and scene rotation for a volume of size N x N x N, where $N = 2^m$, as follows. Order the elements of $\{\mathcal{V}_I\}$ in BTF order. Recursively apply $\mathcal{R}$ to each $\{\mathcal{V}_I\}$ for each element of $\{\mathcal{P}_I\}$ in BTF order and simultaneously to the corresponding $\{\mathcal{V}_O\}$ and $\{\mathcal{P}_O\}$, generating the $\{\mathcal{V}_{I^*}\}$, $\{\mathcal{P}_{I^*}\}$, $\{\mathcal{V}_{O^*}\}$, and $\{\mathcal{P}_{O^*}\}$. Continue the recursion in a like manner until its depth equals m. At the deepest level of recursion, $\mathcal{R}$ provides the mapping from an object space coordinate to its location in image space. Therefore, at the deepest level of recursion, copy the value in the object space array location identified by the object space coordinates to the image space array entry identified by the image space coordinates.

Depending upon the location of P_O in relation to $\{\mathcal{N}_O\}$, two adjustments to the algorithm outlined above must be made at the lowest level of recursion so that the correct image space and object space values are generated. First, because for a front-lower left (FLL) $\{\mathcal{P}_O\}$ rotation is best performed by rotating the entire volume about the origin without first translating the volume center to the origin, negative values for image space coordinates, and hence image array indices, are produced at various rotations. So that the image is actually written into the image output buffer, an image array offset equal to N is added to each image array coordinate. For a sub-cube center (SCC) $\{\mathcal{P}_O\}$, no image array index offset is required. On the other hand, for a SCC $\{\mathcal{P}_O\}$, the algorithm produces the set of coordinates at sub-cube octant 3 and not the sub-cube center at recursion level m-1. This mis-positioning can be corrected by adjusting each of the coordinates generated at level m-1 by applying R twice to $\{\mathcal{V}_{I_3}\}$ at the m-1 level and then subtracting the resultant vector from each coordinate set at level m-1 before the set is sent to level m. The FLL technique does not have this problem as it is designed to produce coordinate values from FLL octant coordinate values and not sub-cube center values.

5.6 The Editing Back-to-Front and Front-to-Back Hidden-Surface Removal Algorithms.

The definition in Section 5.5 highlights an important characteristic of HSR algorithms in general, they do not actually identify hidden-surfaces nor do they remove them. Instead, any voxels in the scene obscured by voxels closer to the observer are either overwritten or ignored in the course of the operation of the algorithm. For example, scanline algorithms must examine each polygon in order to form the correct representation of the volume. The z-buffer and back-to-front algorithms examine all of the primitive data elements in the scene when performing hidden-surface removal, only in these instances the primitive data elements are octrees, voxels, or cuberilles. All of these algorithms produce a display in which the only surfaces displayed are those which are visible from a given point of view. The surfaces which are not visible from a given point of view are removed in the sense that they do not appear in the final scene. Recursive hidden-surface removal algorithms also employ the voxel overwrite[1] strategy.

[1]Strictly speaking, a front-to-back algorithm does not use an overwrite strategy, it uses a prevent-write strategy. However, the effect, as realized in the final image and in terms of computational complexity, is the same.

Recursive hidden-surface removal algorithms usually examine every voxel in order to determine which portions of the volume are visible, thereby insuring that all visible voxels, and hence all visible surfaces, are displayed. However, a useful recursive HSR algorithm for rapid volume editing can be formulated which does not require examination of every voxel. To make the following arguments clearer, the discussion is presented in terms of 2D scenes and objects, but the concepts readily extend to 3D. Refer to Figure 5.18. The figure contains an observer at some distance r_1 from the center of a volume containing an object which has been tessellated into voxels. For the observer to see the visible portion of the object in the figure, examining all the voxels is not necessary since only a fairly small portion of the volume is visible from the indicated position.

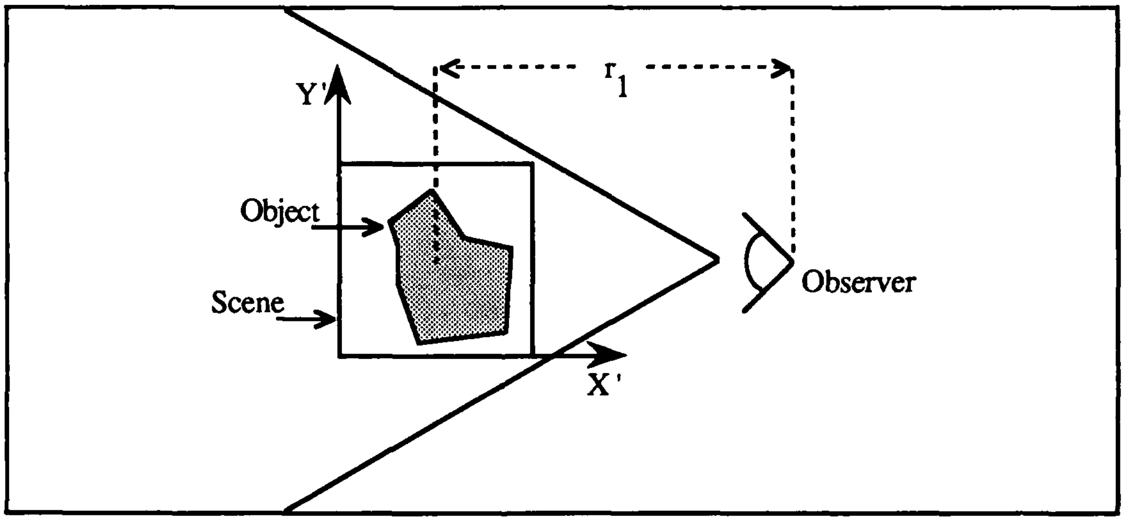


Figure 5.18: Observer Viewing an Object in Image Space.

If the volume is assumed to be solid, then subdivision of the scene provides a means of determining which voxels are hidden. By dividing the volume as shown in Figure 5.19, it is apparent that the quadrants labeled 3 and 0 are not visible by the observer, and so <u>need not be processed</u>. The visible surfaces in the scene are formed from the voxels in quadrants 1 and 2, and thus only the voxels in this quadrant need be examined during the course of hidden-surface removal. As a result, the amount of computation required to generate the scene has been decreased by 50%, albeit at the cost of not being able to form a 3D image. One obvious way to continue to improve the performance of the image rendering procedure is to subdivide quadrants 1 and 2 in the same manner in which the original scene was divided.

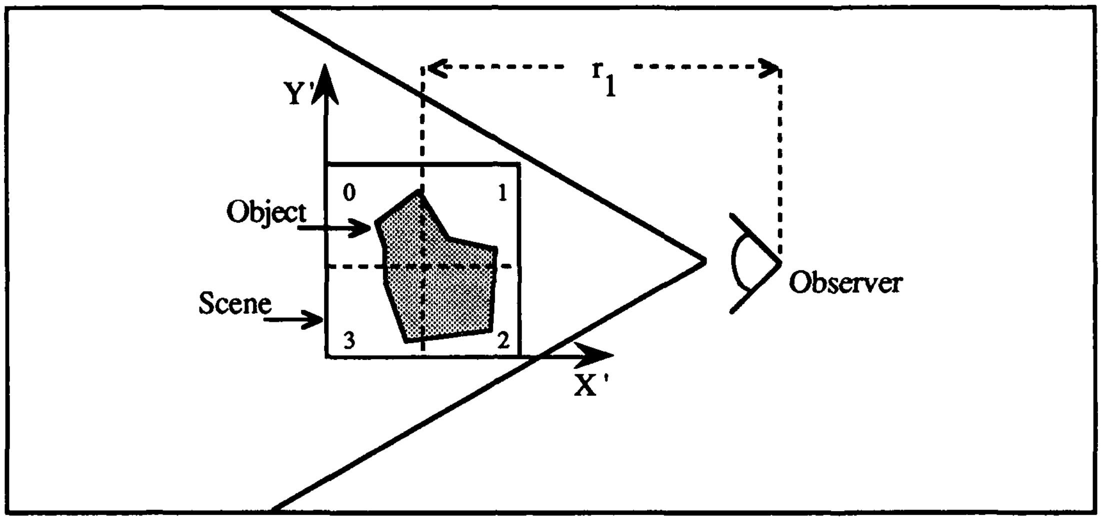


Figure 5.19: Reprise: Observer Viewing an Object in Image Space.

Notice that the rearmost portions of quadrants 1 and 2 are not visible and need not be processed. This second re-sectioning of the scene eliminates half of the voxels in quadrants 1 and 2 from further consideration in the image-rendering operation and reduces the amount of computation required to generate the scene by another 25%. At this point, 75% of the scene has been determined to be hidden from the observer, and not a single voxel has been examined. The process of re-sectioning the remainder of the scene can continue until only those voxels which lie along the front surface of the scene remain, at which time they can be read out to display the final scene.

As described thus far, the re-sectioning algorithm performs hidden-surface removal on a volume more efficiently than the recursive basic hidden-surface removal algorithms used in the SSGs, but has two serious drawbacks. First, the relative positions of the observer and scene are fixed, scene rotation is not allowed. Second, cutting planes cannot be positioned to dissect the volume. Clearly, a more sophisticated approach is required, yet the basic idea of re-sectioning a scene and discarding those portions which do not form visible surfaces remains sound.

The example presented in Figures 5.18 and 5.19 indicates that the key to achieving rapid determination of those portions of the scene which can be discarded is placement of a reference point in a position centered on either the foremost or rearmost face of the volume. Intuitively, it is appealing to place the reference point at the rear of the volume for a BTF algorithm, and at the front of the volume for a FTB algorithm, but either reference point placement is acceptable for either algorithm. The following discussion is based on the intuitive placement approach. The use of a reference point allows the determination of the octants to be cast off to be accomplished using distance comparisons based on the P_I for each octant (and its suboctants) and the reference point. By moving the reference point, called the back-to-front reference point (BTFRP), to a point behind the image space volume[1] and rotating this point in concert with the motion of the observer, a distance test can be used to determine if a suboctant is closer to the back of the volume that its parent octant, and so need not be processed. This change resolves the first objection mentioned above and is employed in both of recursive hidden-surface removal algorithms described below.

The capability for inserting a cutting plane is attained by using two distance tests in image space to ascertain if a set of voxels is intersected by the cutting plane. For the purposes of the following description, the geometry of the tests is illustrated in 2D in Figure 5.20. The assumed current suboctant is octant 3 and its P_I is marked with an X. Therefore, the parent of the assumed current suboctant is the P_I at the scene center, which is also marked with an X. The first test is used to determine if another suboctant lies between the current suboctant and the observer. This test is accomplished by computing the distance from the current suboctant's P_I to the back-of-scene reference point, labeled D_c

[1]Or in front of the volume.

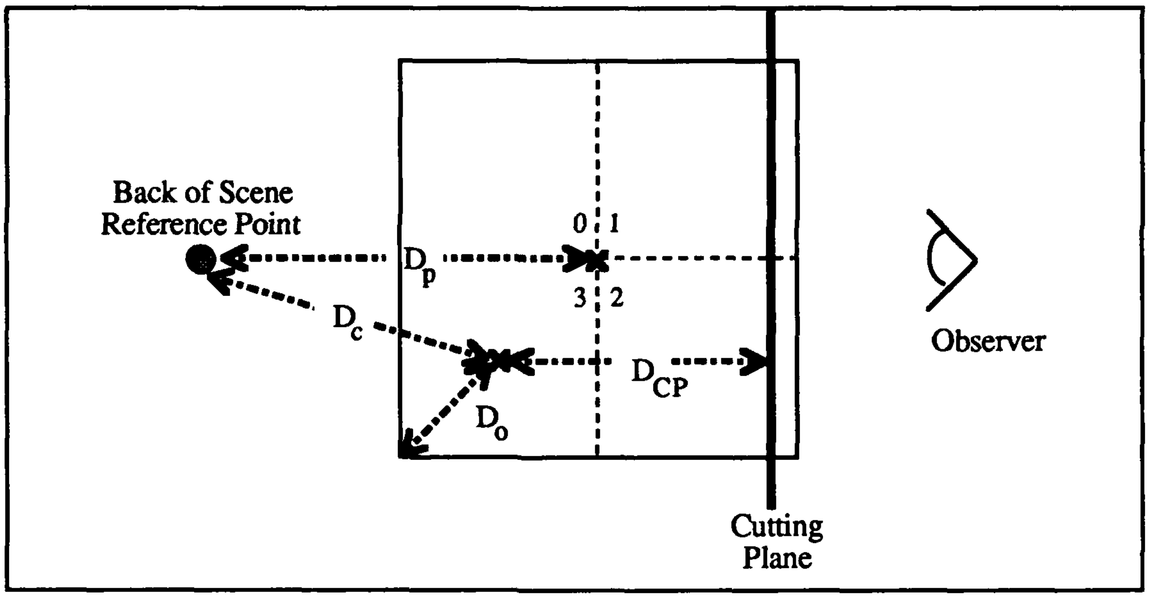


Figure 5.20: Back-to-Front Editing Hidden-Surface Removal Algorithm Geometry.

in Figure 5.20, and comparing this value to the distance from the suboctant's parent's P_I to the back-of-scene reference point, labeled D_p in Figure 5.20. If the parent octant center lies between the suboctant center and the cutting plane, then a sibling suboctant must lie between the current suboctant and the cutting plane, *indicating that the current suboctant is* obscured and so need not be processed.

The second test determines if the cutting plane passes through the current suboctant. This test compares the distance from the current $\{\mathcal{N}_J\}$'s P_I to the cutting plane, labeled D_{cp} in Figure 5.20, against the distance from P_I to the furthest point within its suboctant, which is labeled D_o in Figure 5.20. This test is required because at different orientations, a suboctant can pass the first test and yet contain voxels which intersect the cutting plane. These two tests allow the SSG to cast away substantial portions of its processed volume early in the hidden-surface removal process, thereby reducing the computation performed at the SSG while at the same time permitting arbitrary scene rotation and cutting plane placement. The back-to-front algorithm is described below using the notation introduced in Section 5.5; pseudocode for both the back-to-front and front-to-back algorithms is contained in Appendix E.

Assume an imaged volume of size N x N x N, where $N = 2^m$, and a cutting plane through the volume which is perpendicular to the y-z plane. The origin lies at the front, lower, left of the scene. Divide the imaged volume into eight equal sized octants of size N/2 x N/2 x N/2, producing the $\{\mathcal{N}_0\}$. The back-of-scene reference point is placed in image space at coordinates x=N/2, y=N/2, z=2N[1]. Determine the $\{\mathcal{P}_0\}$ and $\{\mathcal{V}_0\}$ based on whether the algorithm uses the front-lower-left or sub-cube center variation. $\{\mathcal{R}\}$ is as defined previously, and the $\{\mathcal{P}_I\}$ and $\{\mathcal{V}_I\}$ are computed based upon the $\{\mathcal{P}_0\}$ and $\{\mathcal{V}_0\}$.

Let *i* represent octant i of the scene, *ij* represent the suboctant j which lies within octant i, and so on. In image space, let C represent the center of the scene, C_i represent the P_I for octant i, and C_{ij} represent the P_I of suboctant j, which lies within octant i, and so on. The function $D_{rp}(x)$ returns the distance from the point x to the back-of-scene reference point, BTFRP. $D_p(x)$ returns the distance from the point x to the cutting plane in the scene. D_o returns the distance from P_I to the furthest point within the image space suboctant[2]. Perform the following steps <u>in back-to-front order</u> on each of the $\{\mathcal{P}_0\}$. First, determine if the neighborhood around P_I, which is octant *i*, contains data which is visible from the observer's position. The octant is obscured if $D_{rp}(C_i) < D_{rp}(C)$ and the cutting plane does not intersect the octant if $D_p(C_i) > D_o(C_i)$. If these conditions are met, do not process the remainder of octant *i*. If either test is not met, process the remainder of the octant.

Octant *i* now becomes a parent octant of dimension M x M x M, where M=N/4. Now, treat parent octant *i* as though it were the entire scene and determine <u>in back-to-front order</u> which of its eight octants are visible as follows. If the recursion level equals m-1, use $\{\mathcal{R}\}$ and the appropriate coordinate adjustments to generate image space and object

[1]For a front-to-back implementation, the front-of-scene reference point is located at x=N/2, y=N/2,z=-2N.

[2]Letting L be the length of a side of a scene octant, the distance can be computed using the formula: $D_o = \sqrt{(L»(\text{recursion\_level}-1))2 * 3}$ where » is defined as the left shift operator and L = N/2 for the SCC case and N/4 for the FLL case.

space voxel coordinates in back-to-front order and store the image data in the image space array. Otherwise, use $\{\mathcal{R}\}$ to compute the C_{ij} for the next recursion level and then determine which of the new octants *ij* are visible to the observer. The visibility determination is made by computing the distance from the C_{ij} to the cutting plane and the back-of-scene reference point. If $D_{rp}(C_{ij}) < D_{rp}(C_i)$ and $D_p(C_{ij}) > D_o(C_{ij})$ then do not process the remainder of *ij*. If either test fails, consider *ij* to be a parent octant and perform the octant division and distance determination steps for each of the new octants *ijk*. Continue processing smaller and smaller portions of each scene octant *i* until each all suboctants have been either cast away or written to the output buffer. When processing concludes, the two-dimensional slice of the volume along the cutting plane is in the output buffer.

The back-to-front and front-to-back editing recursive HSR algorithms described above were developed by myself prior to the announcement of the editing algorithm employed in the Sun TAAC-1. My editing approach differs from the Sun TAAC algorithm in in two respects. First, it casts away volumes in front of and behind the requested 2D slice instead of computing the slice by following its surface with a Bresenham-like plot as in the Sun TAAC-1. Second, my approach does not depend upon 3D array addressing hardware for its speed and instead relies upon determination of obscured volumes. However, like the approach taken in the Sun TAAC, it is useful for scene editing operations, as it quickly produces the 2D slice of data visible at a given orientation. This allows the user to rapidly orient the scene before spending the time required to accomplish a complete 3D rendering of the volume. The next section presents performance results for the editing recursive HSR algorithm and compares them to the basic recursive HSR algorithm discussed in [illegible]tion 5.4.

5.7 Performance of the Editing Back-to-Front and Front-to-Back Hidden-Surface Removal Algorithms.

Since the editing HSR algorithms described in the preceding section should be faster than the basic HSR algorithms used in Section 5.4, I assessed the editing algorithm's performance within the same suite of packet sizes and interconnection topologies as the basic HSR algorithms. This approach allowed me to verify the conclusions reached in that section concerning packet size and to gather additional data upon which to make an assessment concerning appropriate interconnection topology. The timing results contained within this section are based upon a message packet size of 512 bytes, 4k bytes, 16k bytes,

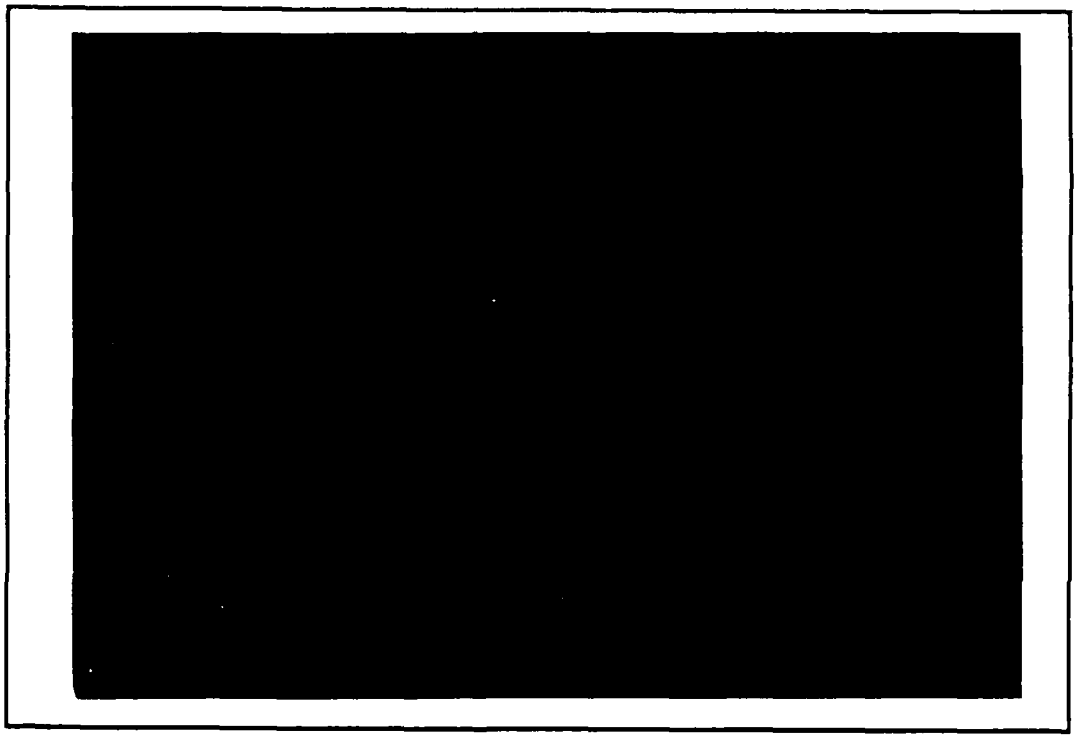


Figure 5.21: Mitre Box Rendered in the Hypercube Interconnection Topology Using the Editing Back-to-Front (BTF) Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 0° z-axis Rotation With the Cutting Plane Positioned at a Depth of Thirty-Three Voxels in Image Space.

and 64k bytes with a scene size of 16^3 voxels at each SSG. Computation and communication times are computed as described in Section 5.3. Using the editing BTF SCC HSR algorithm, a 64^3 voxel volume depicting a human vertebrate in gel was rendered at 0° rotation using the hypercube interconnection topology with 64k byte message packets with a cutting plane placed at a depth of 33 voxels. The resulting image is presented in Figure 5.21.

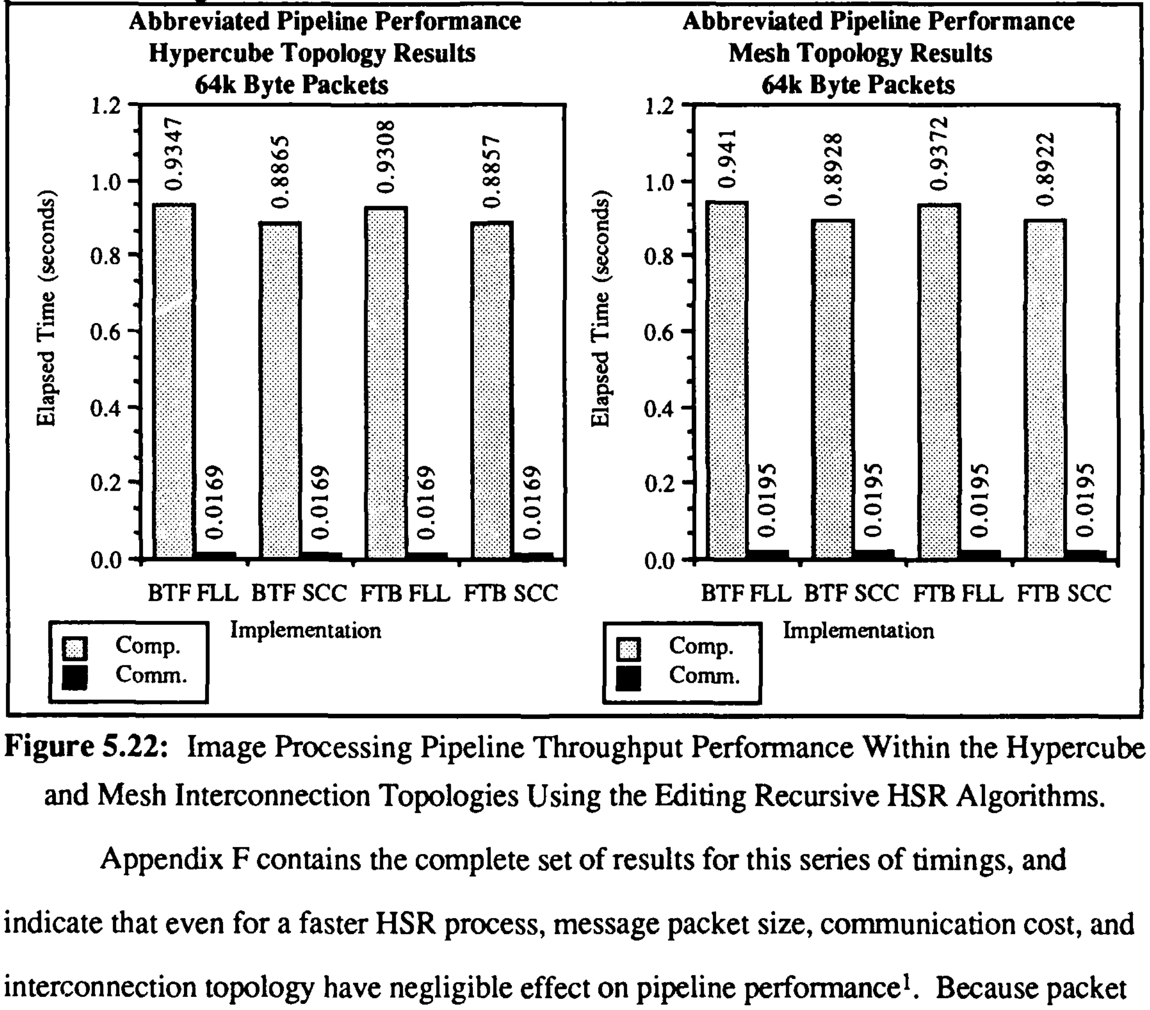


Figure 5.22: Image Processing Pipeline Throughput Performance Within the Hypercube and Mesh Interconnection Topologies Using the Editing Recursive HSR Algorithms.

Appendix F contains the complete set of results for this series of timings, and indicate that even for a faster HSR process, message packet size, communication cost, and interconnection topology have negligible effect on pipeline performance[1]. Because packet size plays such a small part in pipeline performance, the remainder of the discussion in this section can be based upon a single packet size. For consistency with Section 5.4, I chose

[1]Approximately 2% of the elapsed time required to generate an image is spent communicating in a hypercube and 2-3% of the elapsed time is spent communicating in a mesh.

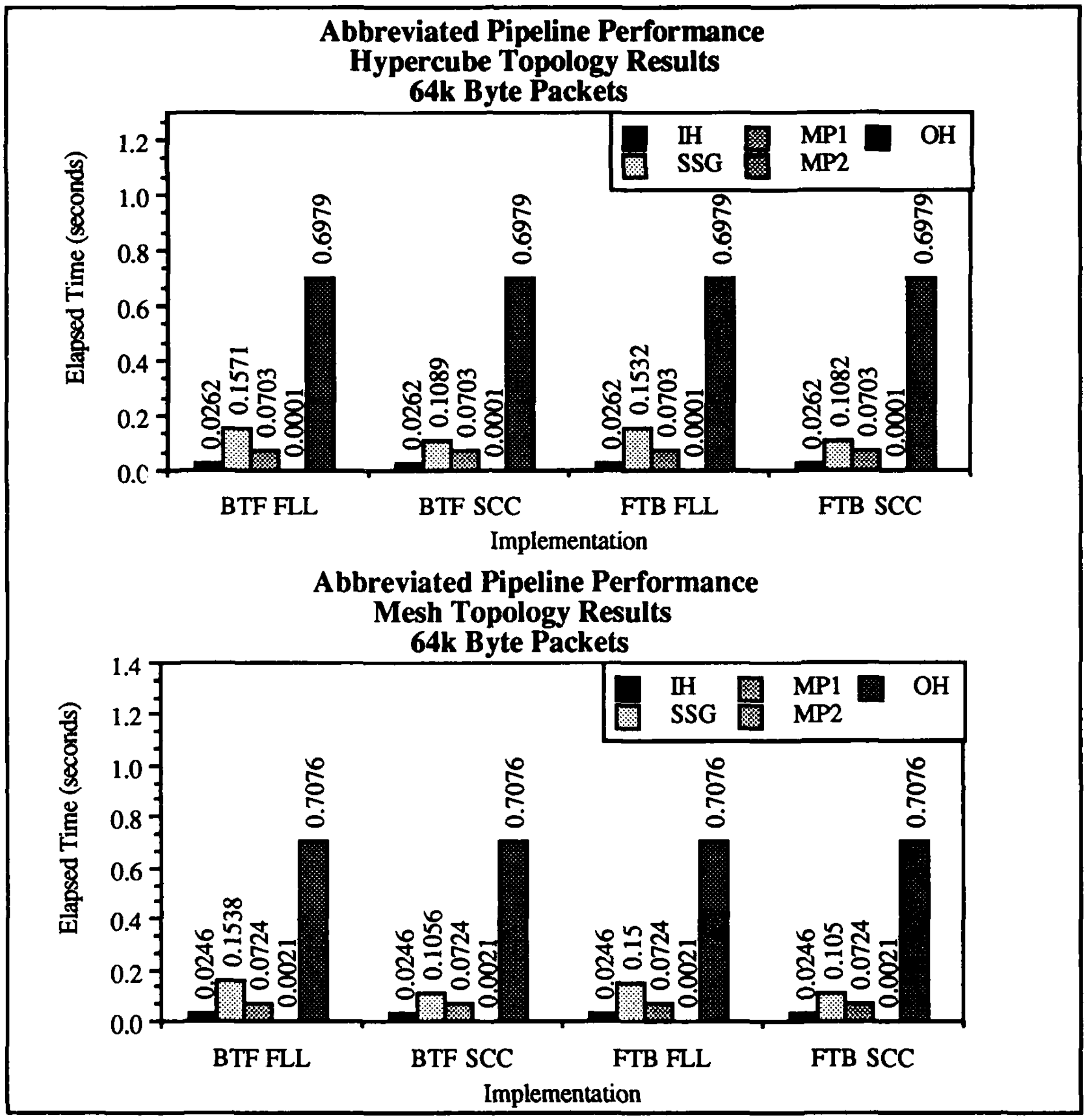


Figure 5.23: Image Processing Pipeline Stage Performance Within the Hypercube and Mesh Interconnection Topologies Using the Editing Recursive HSR Algorithms.

the 64k byte message packets. The throughput results obtained for this packet size on both topologies are presented in Figure 5.22. Individual stage performance results for both interconnection topologies are presented in Figure 5.23.

The results in Figure 5.23 show that T_r for the pipeline is now the OH stage. However, since the operations of this stage can be performed in hardware, a better assessment of the pipeline performance using the editing algorithms is obtained by removing the OH computation time from the pipeline throughput time. The remaining time is the performance of the front-end of the pipeline. This data is presented for all packet sizes on both topologies in Appendix G, with the 64k byte message packet size data also presented in Figure 5.24.

These graphs indicate that computation time is playing a larger role in determining pipeline front-end image generation costs than was at first apparent. In the hypercube topology, communication takes 7-11% of the total image rendering time, depending upon HSR algorithm performance and message packet size. For the mesh topology, the corresponding range is 8-13%. These results have interesting implications for machines

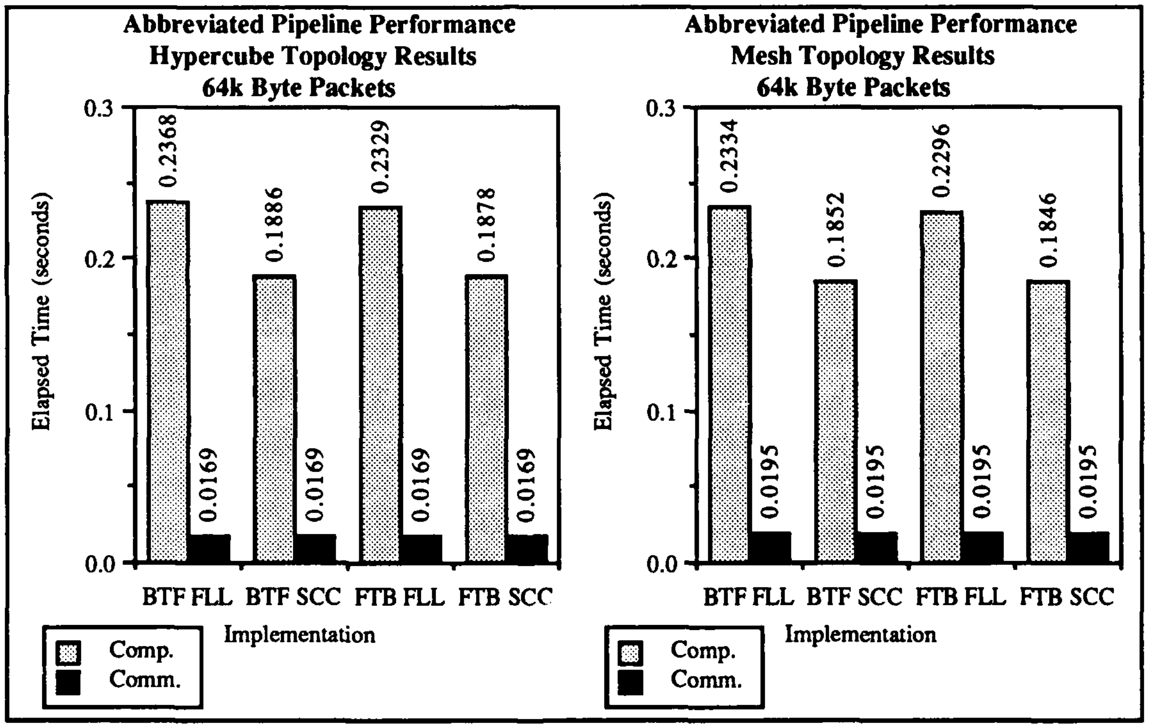


Figure 5.24: Front-End Image Processing Pipeline Throughput Performance Within the Hypercube and Mesh Interconnection Topologies Using the Editing Recursive HSR Algorithms.

with larger numbers of nodes, and this topic is treated in Section 5.8. However, computation time still dominates communication time. With OH functions placed in hardware, the SSG stage determines T_r, and T_e can be approximated by the sum of the SSG and MP_1 stages. The SSG stage remains the performance bottleneck in the pipeline.

Since pipeline performance at larger scene sizes is again determined by SSG performance, the operation of the SSG stage using the editing algorithms on 16^3, 32^3, and 64^3 volumes at an SSG was investigated. These results are summarized in Table 5.3 and graphically in Figure 5.25.

Table 5.3
Elapsed Image Rendering Time Performance of the Editing Recursive Hidden-Surface Removal Algorithms (in seconds).

Supersampled SSG Scene Size	ALGORITHM			
	Back-to-Front, FLL	Back-to-Front, SCC	Front-to-Back, FLL	Front-to-Back, SCC
16^3	.136 seconds	.088 seconds	.132 seconds	.087 seconds
32^3	.549 seconds	.346 seconds	.530 seconds	.340 seconds
64^3	2.215 seconds	1.391 seconds	2.157 seconds	1.367 seconds

As is true for the basic HSR algorithms, the amount of time required by the SSG stage to complete image rendering operation increases linearly with increase in scene dimension. Note that the BTF algorithm now outperforms the FTB algorithm, which is the opposite from the result obtained for the basic algorithm. Whereas for the basic implementation the FLL versions are faster than the SCC versions by 20%, in the case of the editing algorithms the SCC versions are faster than the FLL versions by about 35%. In this case, the difference in performance is attributed to the computation which determines if the cutting plane lies within the volume of the SSG. Recall that this computation determines the point in the volume furthest from P_I and compares this distance to the distance from P_I to the cutting plane. Or, put another way, the test defines a volume which is considered to be the $\{\mathcal{N}_J\}$ volume for the P_I and checks for cutting plane intersection with this volume. In the SCC version, since P_I is centered on $\{\mathcal{N}_J\}$, the volume defined by the furthest point computation is equal to the $\{\mathcal{N}_J\}$ that P_I generates, making this a very

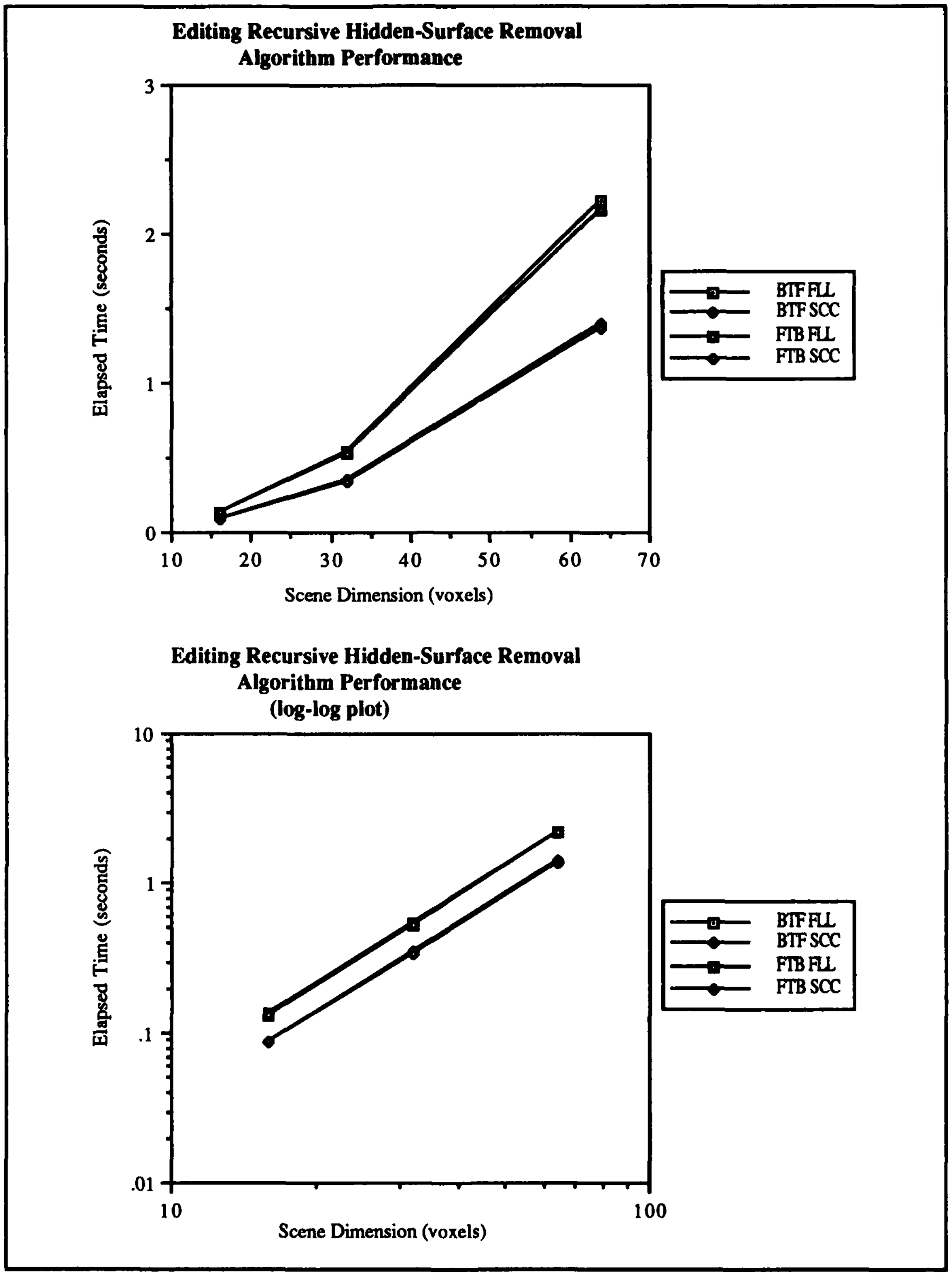
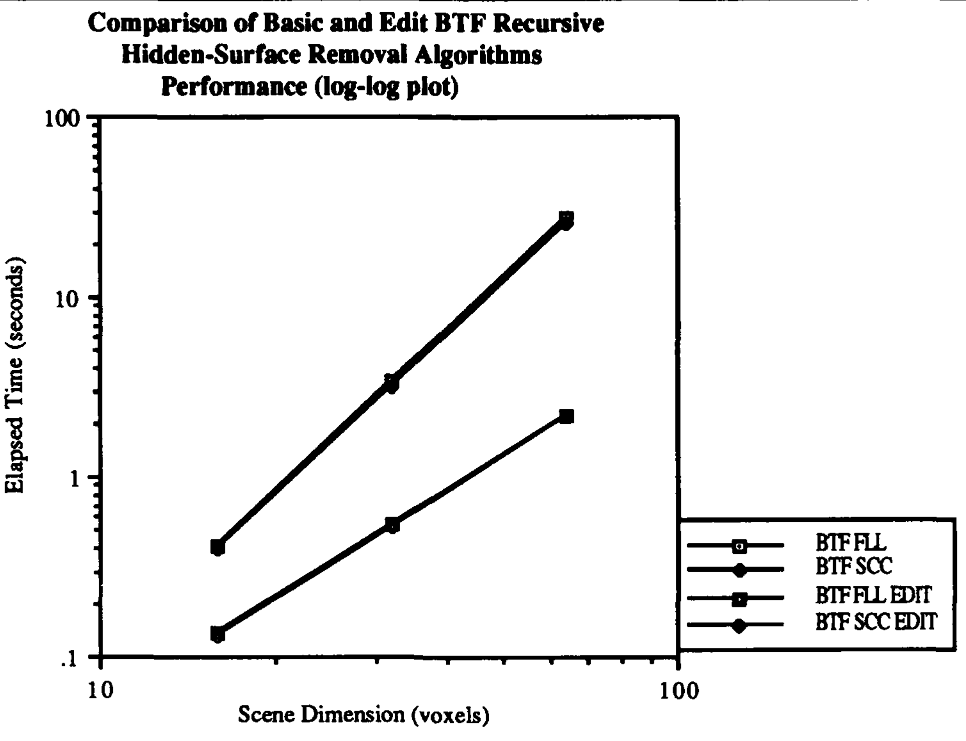


Figure 5.25: Performance of the Editing Recursive Hidden-Surface Removal Algorithms.

precise test for suboctant - cutting plane intersection. In the FLL version, the volume defined in the furthest point computation is equal to eight times the $\{\mathcal{N}_I\}$ volume that P_I generates. The test implicitly treats the P_I as though it is at the center of a volume, rather than the front-lower left, and since P_I and the furthest point lie at opposite points in the volume, a distance equal to the span of the volume is used to test for intersection. So, in effect, the test as used in the FLL version determines at level m in the recursion if the cutting plane intersects the volume at level m-1, making the FLL-based intersection test a less precise test for suboctant - cutting plane intersection than the test used in the SCC version. The upshot of this lack of precision is that suboctants which do not intersect the cutting plane are retained and examined further instead of being ignored, resulting in somewhat poorer performance for the FLL versions than the SCC versions. However, this difference is small when compared to the performance difference between the basic and the editing recursive HSR algorithms.

Figure 5.26 demonstrates that the editing HSR algorithms reduce the amount of time required to generate an image by an order of magnitude as compared to the basic HSR algorithms. Given that the editing algorithms do not generate a 3D image and the manner in which they determine the voxels which lie along the cutting plane, the editing algorithms can be taken as a lower time limit for recursive HSR algorithm performance. Since T_r and T_e are closely tied to SSG performance for editing algorithms, the 3D HSR algorithms presented in the next chapter can be examined without reference to the pipeline, as pipeline image generation time and SSG image generation time are essentially equivalent.

Before turning to the new recursive HSR algorithms presented in Chapter VI, a brief discussion of the relative merits of each interconnection topology is warranted. This discussion is contained in the next section and is followed by the chapter summary.



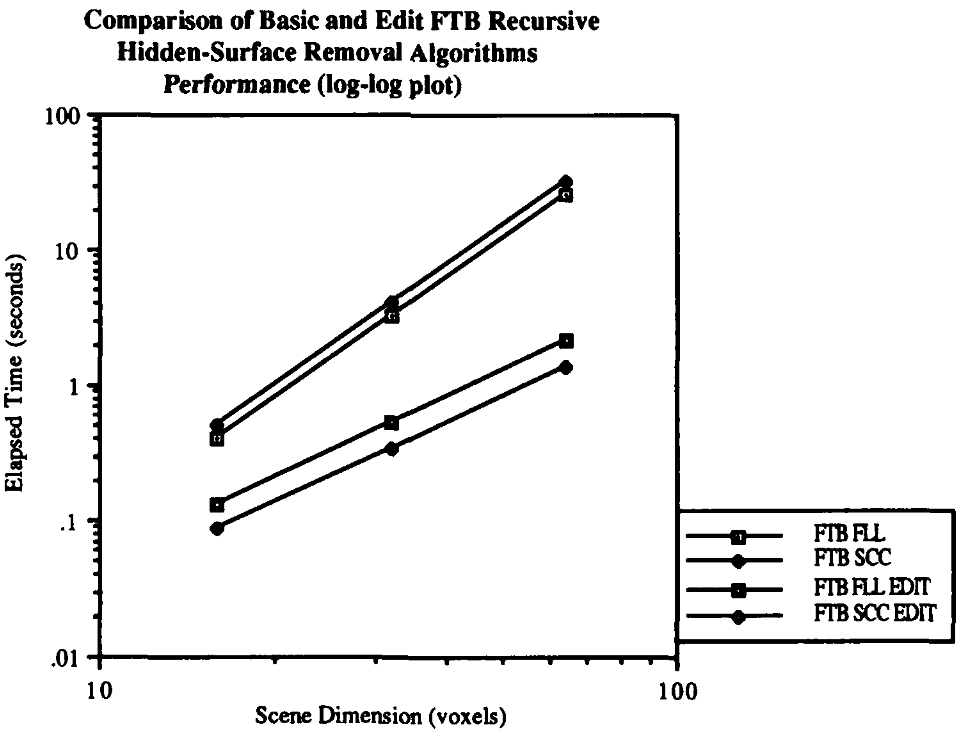


Figure 5.26: Elapsed Image Rendering Time Comparison of the Basic and Editing Recursive HSR Algorithms.

5.8 Comparison of the Multiprocessor Interconnection Topologies.

The results in the preceding section demonstrate that the node interconnection topology and message packet size have marginal impact on the total time required to render an image. That being said, a few additional comments on the relative merits of each topology and overall communication performance factors are in order. First, the packet size should be as large as possible. While any of the four packet sizes is acceptable for distributing user rendering instructions to the processors[1], the use of large message packets reduces the total number of packets used in later pipeline stages when large images are being transported between stages. Second, the hypercube interconnection topology has lower communication cost than the mesh due to its richer node interconnection scheme which provides greater options for packet routing. While this difference is not significant in the performance of the abbreviated pipeline, its effect increases as the number of nodes, and hence the potential for communication "hot spots", in the pipeline increases.

Third, the mesh topology does not require as many processors to accommodate larger image processing pipelines as does the hypercube. The number of nodes in a mesh grows as ceil($\sqrt{N}$), where N is the number of pipeline nodes, whereas the number of hypercube nodes grows with ceil($\log_2 N$). So, for example, whereas a pipeline with 1 IH, 64 SSGs, 9 MPs, and 1 OH requires a 9 x 9 mesh which has 6 unused nodes, the same pipeline requires a dimension 7 hypercube of 128 nodes, 53 of which are unused. So, while the hypercube has lower communication costs than the mesh, this speed is paid for by processors which perform no computation but just serve to route messages. The determination of the correct topology to host the medical imaging pipeline I propose would

[1]Orientation (SCT) packet size is 256 bytes, editing packet size is 72 bytes.

be based on the relative importance of communication speed and processor utilization for a given implementation. Because the hypercube interconnection topology has a richer node interconnection scheme, and hence fewer "hot-spots", than the mesh topology, the hypercube inter-processor communication results scale to larger image processing pipelines. Therefore, when the discussion in Chapter VI turns to a presentation of predicted performance in a multiprocessor architecture, the hypercube interconnection topology is assumed.

5.9 Summary.

This chapter provides a description of the image processing pipeline, data model, coordinate system, multiprocessor interconnection topologies, and timing methodology employed to gather the results presented. The performance of the image processing pipeline using the basic BTF/FTB recursive HSR algorithms was examined, revealing that the hidden-surface removal operation within the SSG stage of the pipeline is the performance bottleneck. Prior to describing a BTF and FTB algorithm suitable for rapid scene editing, a definition for the class of recursive HSR algorithms was introduced. Based on the definition and an examination of the operation of most hidden-surface removal algorithms, modifications to the BTF and FTB algorithms which perform rapid 2D renderings of the portion of the volume along the cutting plane were developed. These modifications reduce the time required to generate an image by an order of magnitude, although the capability to form 3D images is sacrificed. These editing algorithms are useful because they permit swift inspection of the scene using different display criteria. Once the set of desired viewpoints and parameters is determined, then the time consuming 3D image rendering may be performed, but only on the desired set. The payoff for this approach is substantial savings in the amount of time spent examining the medical image volume. A comparison of the two multiprocessor interconnection topologies revealed that neither topology is clearly superior and that communication costs are a minor factor in determining pipeline throughput. Because the hypercube interconnection topology scales cleanly to large multiprocessor implementations, the hypercube interconnection topology is the assumed topology in the next chapter.

Having determined a method for scene editing, the 3D image production SSG bottleneck remains. The next chapter presents back-to-front and front-to-back recursive hidden-surface removal algorithms which reduce the magnitude of the performance bottleneck by building upon both the distance test concept and the recursive HSR algorithm definition introduced in this chapter.

CHAPTER VI

DYNAMIC ADAPTIVE HIDDEN-SURFACE REMOVAL

6.1 Introduction.

Chapter V presented a discussion of the interconnection topologies, the performance of the basic recursive back-to-front (BTF) and front-to-back (FTB) algorithms within these topologies, and introduced two new scene editing recursive FTB and BTF algorithms. This chapter extends this work through an investigation of a new strategy for hidden-surface removal, dynamic algorithm selection. This approach, coupled with improvements to the basic FTB and BTF algorithms, results in computational savings at each Sub-Scene Generator (SSG) node in the medical imaging pipeline described in Chapter V.

Section two reviews the discussion of the performance of the medical imaging pipeline presented in Chapter V. Section three examines hidden-surface removal (HSR) algorithms, with particular attention toward their suitability for dynamic algorithm selection, and previous approaches to hidden-surface removal. Section four describes the adaptive recursive BTF HSR algorithm and section five describes the adaptive recursive FTB HSR algorithm. Section six describes the dynamic hidden-surface removal process used in the SSG stage and its performance within the medical image processing pipeline. As Chapter V demonstrated that communication costs and interconnection topology are minor performance issues, these concerns are only briefly discussed in this section. Section seven contains the chapter summary.

6.2 Medical Imaging Pipeline Performance Considerations.

The medical imaging pipeline used in this chapter is the testbed medical imaging pipeline depicted in Figure 6.1. Its operation is described in depth in Chapter V.

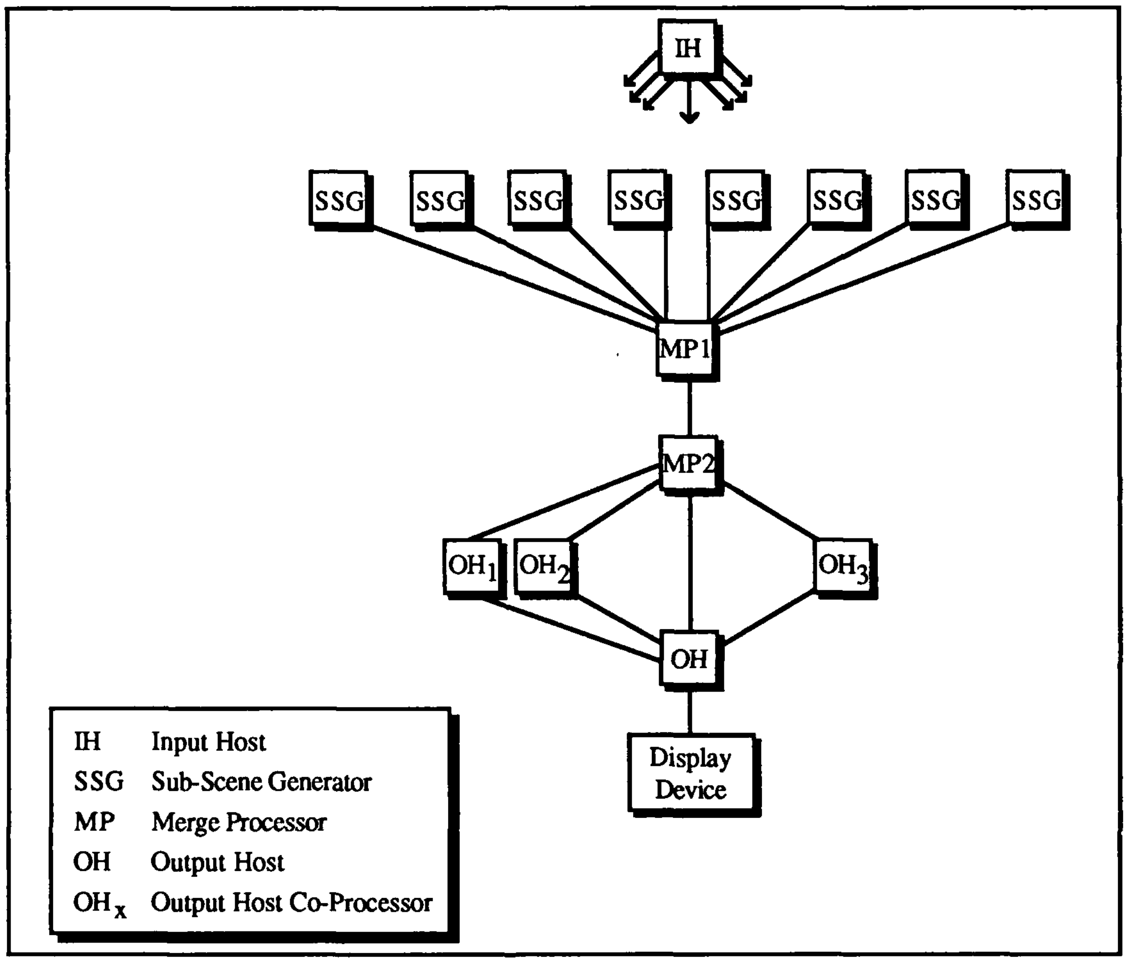


Figure 6.1: Abbreviated Image Processing Pipeline.

The medical imaging pipeline shown in Figure 6.1 consists of five stages, Input Host (IH) stage, the SSG stage, the Merge Processor (MP) stage(s), the Output Host (OH) stage, and image display. The IH accepts user inputs and broadcasts these commands to the rest of the pipeline. The SSGs perform hidden-surface removal, clipping, rotation, anti-aliasing and thresholding on their individual cubes of data. The

MPs accept data from the preceding stage and merge it into a single image which is then forwarded to the next stage. The OH takes the output from the final MP, shades it, and writes this information to the frame buffer for display. As demonstrated in the preceding chapter, the SSG stage is the performance bottleneck for the pipeline, requiring roughly 2/3 of the total time required to render the image. Because the rate at which images emerge from the pipeline, T_r, is determined by the pipeline stage with the worst performance, improvement in the performance of the SSG stage translates directly to improved pipeline performance. In fact, as shown in Chapter V, pipeline performance can be closely approximated by SSG performance and interconnection topology and communication costs have negligible effect on pipeline performance.

Chapter V described the first attempts to reduce the bottleneck using two new HSR algorithms, one operating from front-to-back and the other operating from back-to-front, which significantly improved the performance of the pipeline by modeling the data volume at each SSG as a solid cube. These results were unsatisfactory, however, because the resulting images are only two-dimensional (2D) slices of the volume at each SSG rather than being full three-dimensional (3D) renderings. These algorithms are appropriate for scene editing purposes, but do not generate 3D images.

The bottleneck in the SSG stage is caused by two factors, the requirement that the entire data volume at each SSG be traversed when generating the image and the volume of data operated upon when performing anti-aliasing. The traversal requirement can be eliminated by employing an algorithm which determines if the current state of the image is complete, that is, additional computation does not add additional information to the image. The requirement to add a capability for determining when the image is complete in the SSG stage led to the development of the algorithms discussed below in sections 4 and 5. Before turning to these new approaches, previous HSR techniques are reviewed in the next section with a view toward assessing their amenability for accelerating the image generation process by adaptive termination.

6.3 Hidden-Surface Removal - Options for Reducing Computation.

This section expands upon the discussion of the various approaches to the hidden-surface removal problem presented in section 3.10. The discussion in section 3.10 is not repeated here, instead each class of HSR algorithms is assessed for its ability to be modified to reduce image rendering time. The section concludes with the rationale for continued development of the recursive BTF/FTB algorithms.

Depth-sort algorithms have witnessed little use in medical imaging applications. Since medical images are inherently spatially pre-sorted, no explicit sort is required to arrange the scene elements into BTF order. I did not consider depth-sort algorithms to be a practical approach to reducing computation time in the SSG stage.

The most common approach to HSR is the use of the z-buffer algorithm. Previous efforts to reduce running time for this algorithm focused on the hardware involved. Booth, in [Boo86], describes hardware modifications which provide for overlap of image rendering and image display operations, using additional bits in the z-buffer to indicate which frame buffer contains the currently displayed image and currently rendered image. This approach achieves some speedup through overlapped operation and the use of image display hardware to reset the z' information in the z-buffer before the next image is generated into it. Other hardware approaches, as in the Titan, Stellar, and Pixel Machine, make use of video RAM (VRAM) which support faster access to z-buffer memory for writes and reads than DRAM. While these hardware techniques decrease the image rendering time for an implementation, no fundamental improvement to the z-buffer algorithm has been made. The likelihood of making additional algorithmic improvements to this algorithm is slight, because the advantage of using a z-buffer algorithm is its ability to process arbitrary writes to image space with guaranteed correct HSR. Since algorithmic improvements to HSR performance require ordered, not random, movement through image space in order to determine when the image is complete, and z-buffer algorithms are designed to eliminate the requirement for establishing a processing ordering for the scene, I did not consider z-buffer algorithms suitable for further investigation at this time. The capability of the z-buffer algorithm to achieve correct HSR for an image even though the image elements are processed in random order, while important in other image rendering applications, offers no advantage in medical imaging as the data are spatially pre-sorted.

The third class of HSR algorithms perform either back-to-front (BTF) readout of the scene or front-to-back (FTB) readout without overwrite. The basic BTF algorithms do not use any type of decision mechanisr ɔ limit image generation processing for the same reasons ascribed to the z-buffer algorithm. However, as shown in Chapter V, a distance check can be used to determine when the cutting plane does not intersect an octant, thereby reducing the computations performed when rendering an image. An alternative form of the edit BTF algorithm, called the adaptive BTF algorithm, which renders 3D images at reduced computational cost is described in this chapter.

A front-to-back algorithm, on the other hand, can check after each screen space write to determine if all the image space locations are filled in, allowing the algorithm to stop when all of screen space is filled. However, since a straightforward approach requires a check for each write to screen space, the magnitude of the reduction in image generation time questionable. There has been one reported FTB algorithm which does limit the number of voxels accessed and transformed when rendering an image, the Dynamic Screen Technique described in [Rey87] and Section 3.10.7. This algorithm requires a preprocessing step which transforms the gray-scale voxel representation of the volume into a binary line-segment description. The algorithm operates by writing the line-segments into the dynamic sc n in front-to-back order and terminates when the screen is full. As each segment is written to the screen, the associated voxel data is written to the frame buffer. This approach does reduce the amount of time required for rendering the image, but the requirement for a pre-processing step increases the total time required for image generation, and has the same problem of separating image from background associated with it as any other image segmentation technique. This chapter describes a new FTB algorithm which operates directly upon the gray-scale image and employs adaptive algorithm termination.

The definition of recursive BTF/FTB algorithms presented in the previous chapter indicates how a termination point for the FTB and BTF algorithms can be specified. A basic characteristic of recursive HSR algorithms is their property of volume coherence. This algorithmic characteristic allows one to make decisions concerning the processing of volumes based on the property of a single point within the volume. This property, while not explicitly mentioned in the definition, is the key property of the octree model and associated algorithms because these algorithms characterize a volume by determining homogeneous sub-volumes and then operating on these volumes as though they were a single point. The recursive BTF and FTB HSR algorithms, because they move through space octant-wise, permit each octant and suboctant to be characterized based on its location in the volume relative to the observer and other external reference points. This property can be exploited as follows. Since both the FTB and BTF algorithms process object space by recursive inspection of its sub-volumes, a check at the point where each algorithm proceeds to a new sub-volume can determine if the sub-volume contributes additional information to the final image and must be processed. The two new adaptive termination algorithms which exploit this property for HSR are described in the next two sections.

6.4 The Adaptive Recursive Back-to-Front Hidden-Surface Removal Algorithm.

Given the volume coherence property possessed by recursive BTF/FTB algorithms, and that this property can be exploited when a new set of suboctants is generated, all that remains to be determined is an appropriate method for deciding when the BTF/FTB algorithm should cease processing an octant. For a BTF algorithm, this determination is made using a distance computation strategy similar to that employed in the editing algorithms, albeit with one significant difference. In order to generate a 3D image, the algorithm must process all image space octants intersecting or lying behind the cutting plane and all the remaining image space octants can be safely ignored. Therefore, the adaptive recursive BTF algorithm processes an object space octant only if the cutting plane lies within or in front of the corresponding image space octant. This test permits the algorithm to terminate operation when all of the image space octants behind-of or intersecting the cutting plane have been processed. This strategy prevents the SSGs in the medical imaging pipeline from examining those octants which lie in front of the cutting plane, leading to a computational saving. Using thr notation in Chapter V, the adaptive recursive BTF algorithm is described for the subcube-center (SCC) case in the following paragraphs.

The adaptive BTF hidden-surface removal algorithm uses the same vector sets $\{\mathcal{V}_O\}$ and $\{\mathcal{V}_I\}$, the same dimension object space N x N x N, where $N = 2^m$, and the same $\mathcal{R}$ as the recursive BTF algorithm described in Chapter V. The z'-value for each voxel location in image space is also generated and can be stored for later use by shading hardware/software. Define *i* to represent octant *i* of the scene, *ij* to represent suboctant *j* within octant *i*, and so on. In image space, let C represent the center of the scene, C_i represent the P_I for octant $_i$, and C_{ij} represent P_I of suboctant *ij*. $D_{rp}(x)$ returns the distance in image space from the back-to-front reference plane to the point x, $D_p(x)$ returns the distance in image space between the point x and the scene's cutting plane and D_o returns the distance from P_o to the furthest point within the suboctant[1].

The volume is first rotated to the desired orientation, and a cutting plane (CP), defined by the equation $Ax + By + Cz + D = 0$, is inserted into the image space volume

[1] Letting L be the length of a side of a scene octant, the distance can be computed using the formula: $D_o = \sqrt{(L \gg (\text{recursion\_level-1}))^2 * 3}$ where » is defined as the left shift operator and L = N/2 for the SCC case and N/4 for the FLL case.

such that the plane is parallel to the x'-y' plane. The back-to-front reference plane[1] (BTFRP) is defined by $Ax + By + Cz - \Delta = 0$ such that $\Delta >> N$ ($\Delta = 5N$ works well) and lies behind the observer. This definition places the BTFRP parallel to the cutting plane in image space. Using these two planes provides the capability to determine if an octant lies closer to the BTFRP (and hence the observer) than does the cutting plane. This calculation is performed because if an octant lies closer to the BTFRP than the cutting plane, and the cutting plane does not intersect the octant, the octant can not contribute to the final scene and so need not be examined. The geometry of these distance tests, which are described next, is portrayed in 2D in Figure 6.2.

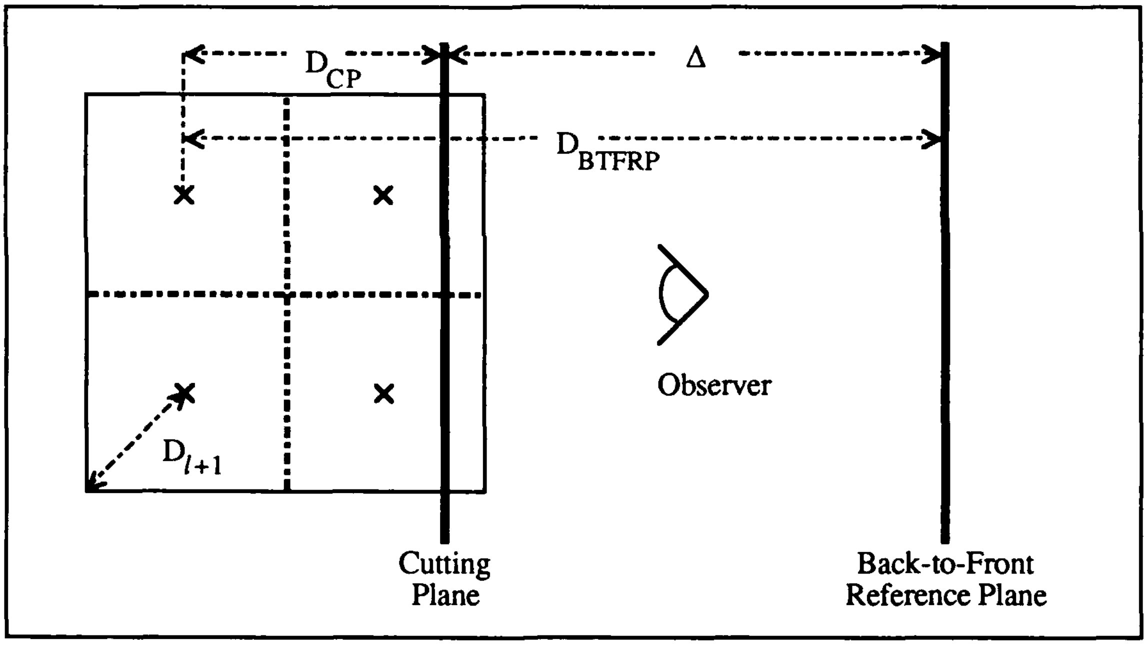


Figure 6.2: Adaptive Recursive Back-to-Front (BTF) Hidden-Surface Removal Algorithm Image Space Geometry.

Having established the rotation of the scene, cutting plane position, and BTFRP position, the coordinates for successively smaller octants in image and object space are

[1] A reference plane must be used, and not a point, because the cutting plane distance test determines the perpendicular, shortest, distance from P_0 to the cutting plane. The use of a back-to-front reference point would mistakenly retain octants for examination which should be eliminated from the HSR processing.

generated in BTF order as described in Chapter V. The difference in operation between the adaptive recursive HSR algorithm and the basic recursive HSR algorithm occurs when a new suboctant's coordinates are generated. At this point, the position of the suboctant relative to the cutting plane is determined, with the results of this test determining if the object space to image space transformation is performed. The test requires first determining the distance from the suboctant center to the cutting plane, $D_{CP} = D_p(C_i)$, and the distance from the suboctant center to the BTFRP, $D_{BTFRP} = D_{rp}(C_i)$. If $D_{CP} > D_o(C_i)$ and $D_{BTFRP} < \Delta$, do not process the corresponding object space suboctant because the image space suboctant lies closer to the BTFRP than the cutting plane[1] and the cutting plane does not lie within the bounds of the octant. If either condition fails, process the suboctant because either the cutting plane lies within the suboctant or the suboctant is further from the BTFRP than the cutting plane.

Continuing in the same manner, treat parent octant *i* as though it were the entire scene and determine the visibility of its eight suboctants. If the recursion level equals m-1, use $\{\mathcal{R}\}$ and the appropriate coordinate adjustments to generate image space and object space voxel coordinates in back-to-front order and store the image data in the image space array. Otherwise, use $\{\mathcal{R}\}$ to compute the C_{ij} for the next recursion level and then determine which of the new octants *ij* are visible to the observer. Compute the required distances, the distance from the suboctant center to the cutting plane, $D_{CP} = D_p(C_i)$, and the distance from the suboctant center to the BTFRP, $D_{BTFRP} = D_{rp}(C_i)$. If $D_{CP} > D_o(C_{ij})$ and $D_{BTFRP} < \Delta$ then do not process the remainder of *ij*. If either test fails, consider *ij* to be a parent octant and continue with the BTF HSR process, and the associated distance determination steps, for each of the new suboctants *ijk*. Continue processing the successively smaller suboctants and applying the distance tests until the entire volume is examined. The computational savings realized using this algorithm is illustrated in 2D in

[1] Hence, the sub-octant lies in front of the cutting plane.

figure 6.3, where the octants the adaptive BTF HSR algorithm avoids examining are shaded. The procedure using the front-lower-left (FLL) based algorithm is identical to that described above.

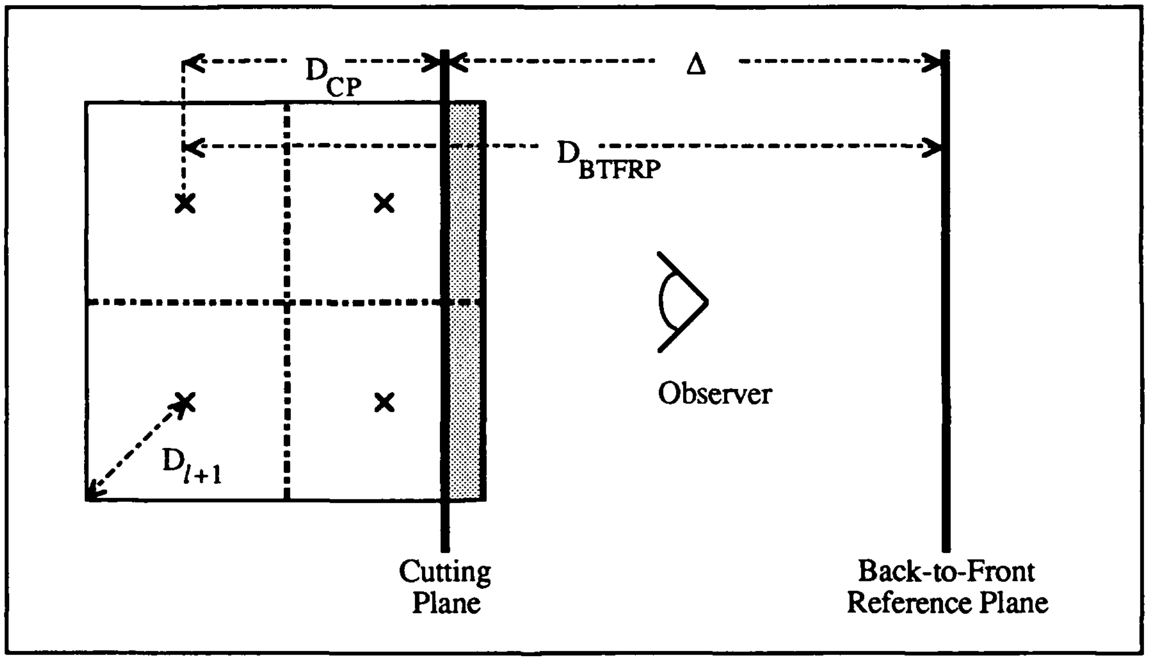


Figure 6.3: Processed Volume Savings Realized Using the Adaptive Recursive Back-to-Front (BTF) Hidden-Surface Removal Algorithm.

An additional algorithmic improvement, object space pre-generation, can be made to increase the speed with which the image is rendered. This improvement alters the way the voxel data is stored and eliminates object space coordinate generation by storing the voxel data within an oct-tree. The value of oct-trees[1] for computational savings has been previously demonstrated in the ray-tracing environment by Fujimoto ([Fuj86])[2], Glassner

[1]Oct-tree is used instead of the conventional octree to emphasize that the tree does not identify homogeneous volumes and assign the resultant density value to leaf nodes, as in Meagher's octree. Instead, each voxel is represented by a leaf and no attempt is made to locate homogeneous volumes. The oct-tree is a hierarchical encapsulation of the spatial relationships between voxels in a volume.

[2]Fujimoto et. al. call their structure a Spatially Enumerated Auxiliary Data Structure (SEADS) and use it to store the parameter descriptions of the object(s) which intersect the terminal voxels volumes.

([Gla84])[1], and Levoy ([Lev89a], [Lev89b])[2], although the approach described here was developed independently of their efforts. Levoy's work is the only previous work which, to my knowledge, uses an oct-tree structure to capture the spatial relationships between voxels. The strategy and operation of the object space pre-generation optimization is described next.

The motivation for the adoption of the oct-tree structure comes from an observation concerning the manner in which the recursive BTF algorithm operates. When computing object space coordinates, the same values are generated for every rendering because $\{\mathcal{R}\}$, $\{\mathcal{V}_0\}$, and $\{\mathcal{P}_0\}$ do not change, therefore the only difference in coordinate values between renderings is the order in which coordinate values are produced. Consequently, there is no need to generate the coordinates anew for each display so long as the requisite spatial relationships are captured within a data structure. Object space pre-generation captures these spatial relationships, thereby allowing the object space coordinates to be computed once. In order to minimize the SSG memory requirement, the 3D object space array is eliminated and the object space oct-tree leaf nodes hold individual voxel values. Image rendering is performed by traversing the oct-tree in BTF order while generating the image space coordinates in parallel with the oct-tree traversal, thereby de-coupling object space and image space coordinate computations. I should note that object space pre-generation, while efficacious in a distributed-memory multiprocessor, may not be useful in a single processor environment because of the large amount of memory required to hold the entire oct-tree for a typical medical image. The use of an oct-tree as described here to capture the density values and spatial relationships is effective for two reasons. First, the distributed memory architecture of the assumed MIMD machine allows the oct-tree to remain within

[1]Glassner's octree contains a list-based description of the objects within the terminal voxels and a hash table is used for leaf access rather than tree traversal.

[2]Levoy terms his structure a hierarchical dataset enumeration, or, alternatively, a pyramid. Each leaf of his oct-tree structure contains a single voxel value.

memory at all times, therefore paging processing overhead is not incurred and the object space computation reduces to merely following pointers. Second, the partitioning of object space among the SSGs ensures a shallow oct-tree depth at all SSGs, therefore the oct-tree memory requirement at each SSG is relatively low.

The oct-tree is constructed as described above to hold the N x N x N ($N=2^m$) normal-resolution object space. During image rendering, when a terminal oct-tree node value is generated its value is checked against the user-defined density-threshold, and if it passes this test then the voxel density value is passed to the next-deeper level of recursion. At this, the m+1 level of recursion, the eight image-space voxel coordinate values are

Figure 6.4: Human Vertebrate in Gel Rendered in the Hypercube Interconnection Topology Using the Adaptive Back-to-Front (BTF) Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 0° z-axis Rotation and σ = 100%.

computed and the single object space density value is written to the image array eight times, thereby generating a supersampled image.

Pseudocode for the complete algorithm, including the object space pre-generation optimization, is presented in Appendix H.

The image rendering accuracy assessment methodology described in section 5.2.4 demonstrates that the adaptive BTF HSR algorithm generates correct images. Using the adaptive recursive BTF sub-cube center (SCC) HSR algorithm, a 64^3 voxel volume depicting a human vertebrate in gel was rendered using the hypercube interconnection topology with 64k byte message packets at 0° z-axis rotation. The resulting image is presented in Figure 6.4.

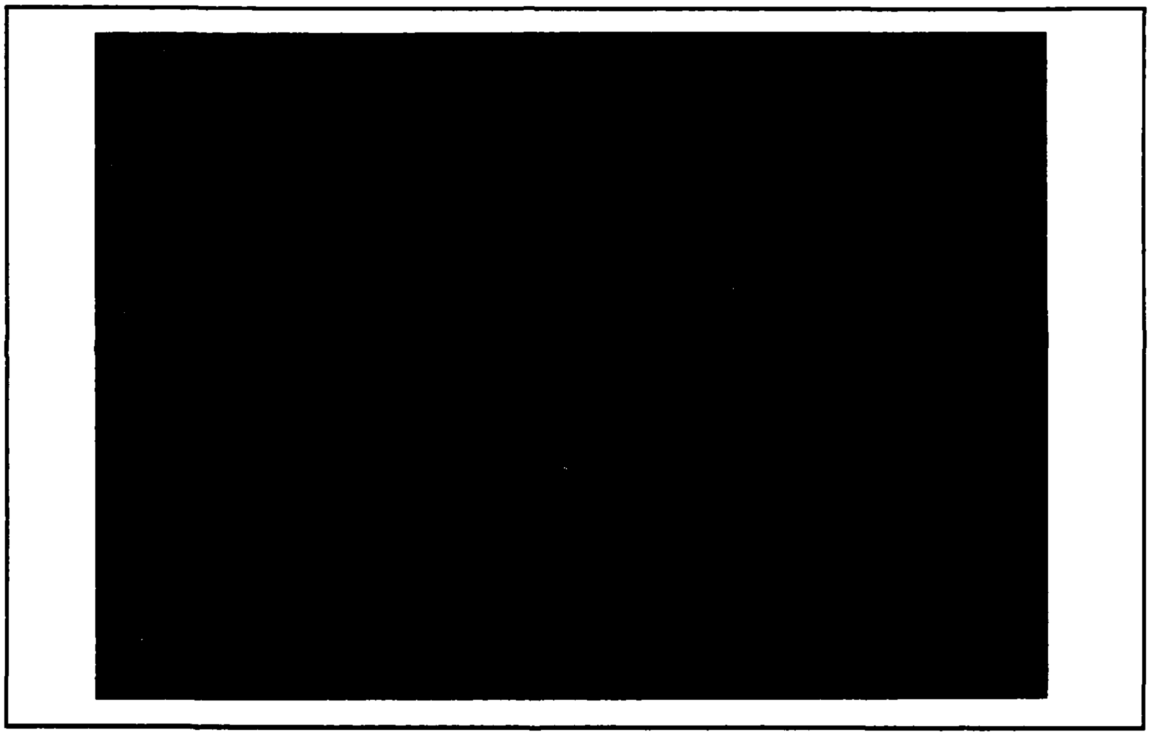


Figure 6.5: Mitre Box Rendered in the Hypercube Interconnection Topology Using the Adaptive Back-to-Front (BTF) Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 0° z-axis Rotation and $\sigma = 100\%$.

Using the Adaptive Recursive BTF SCC HSR algorithm, a 64^3 voxel volume depicting a mitre box was rendered using the hypercube interconnection topology with 64k byte message packets at 0° rotation as well as 45° x-axis and 45° y-axis rotation. The resulting images are in Figures 6.5 and 6.6.

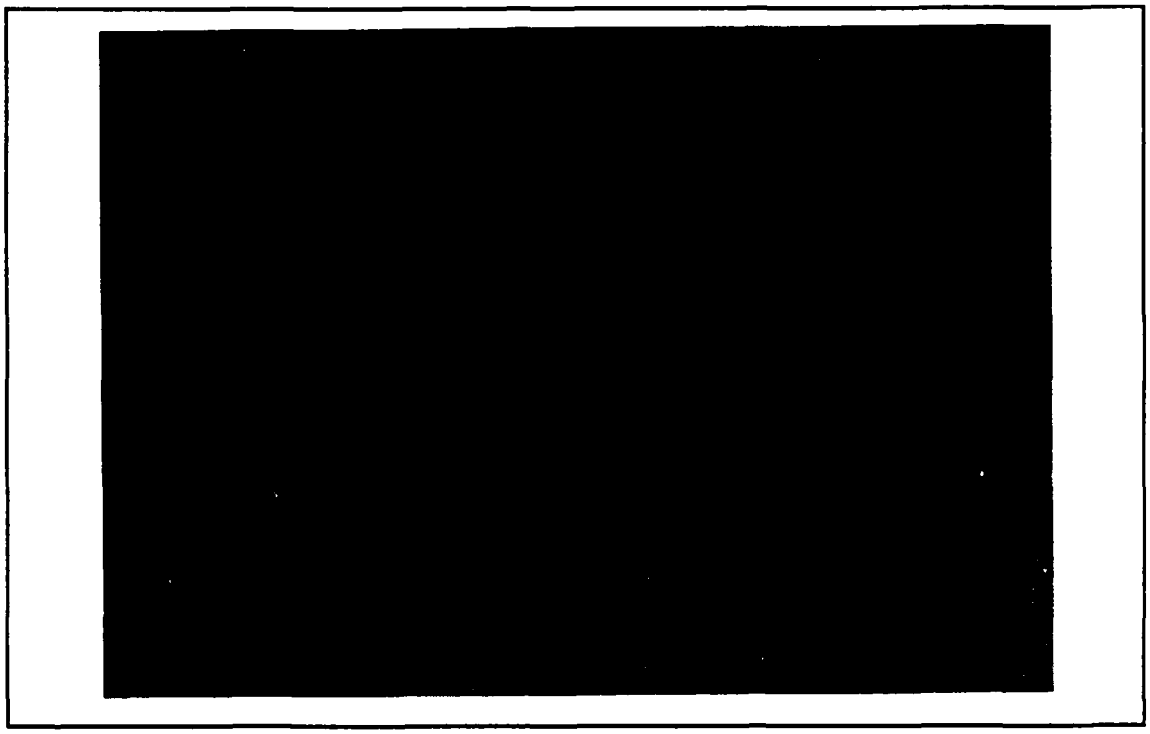


Figure 6.6: Mitre Box Rendered in the Hypercube Interconnection Topology Using the Adaptive Back-to-Front (BTF) Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 45° x-axis and 45° y-axis Rotation and $\sigma = 100\%$.

Using the timing methodology described in section 5.2.4, the performance of the adaptive BTF HSR algorithm was determined. As demonstrated in Chapter V, the pipeline's performance can be approximated by the SSC's performance, so the discussion is limited to the performance of the adaptive BTF HSR algorithm operating there. Because the amount of time required to perform HSR in the SSG stage should decrease as the cutting plane moves deeper into the image space volume at each SSG and as the threshold window saturation value[1], σ, decreases, the adaptive recursive BTF HSR algorithm performance was examined along several dimensions. The experiments examined the performance of the algorithm for a range of image space cutting plane depths, scene sizes,

[1]The threshold window saturation, σ, is the percentage of the voxels within $\{\mathcal{N}_o\}$ which fall within the threshold window set by the user.

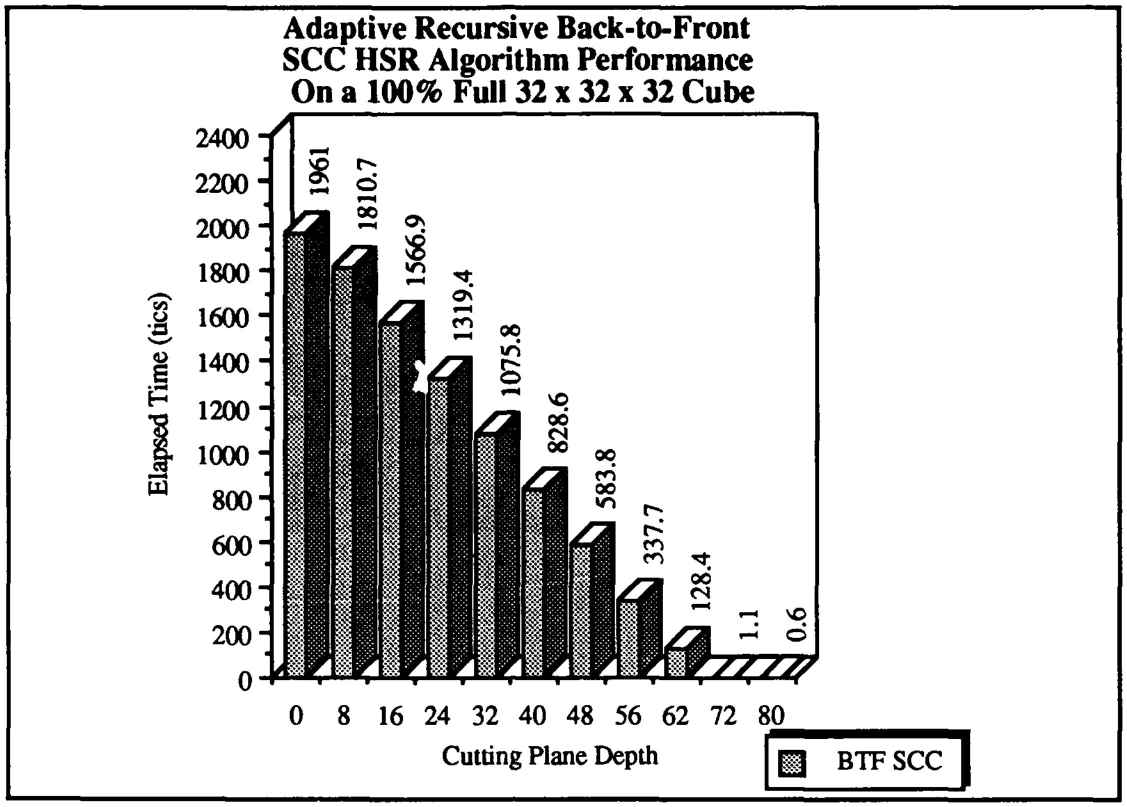


Figure 6.7: Performance of the Subcube-Center (SCC) Variant of the Adaptive Recursive Back-to-Front (BTF) HSR Algorithm with $\sigma = 100\%$.

and σ values. These results are presented graphically in Figures 6.7 through 6.10 for both the FLL and SCC variants of the algorithm on 32^3 object space voxel volumes[1] for $\sigma = 10\%$ and $\sigma = 100\%$[2]. Complete results for both variants for 8^3, 16^3, and 32^3 voxel object space volumes across a range of σ values are summarized in Appendix I.

A comparison of the performance of the FLL and SCC variants of the basic, scene editing, and adaptive BTF HSR algorithms is presented in Figures 6.11 and 6.12 for an object space volume of 32^3 voxels in object space across a range of threshold window

---

[1]Object space values are used to indicate the size of the object space oct-tree and for consistency with the specification of image sizes in Chapter V. The rendered image space volume is eight times the object space volume.

[2]In these and subsequent graphs, the percent full figure refers to the saturation of the object space volume at the SSG and not the saturation of the portion of screen space at the SSG.

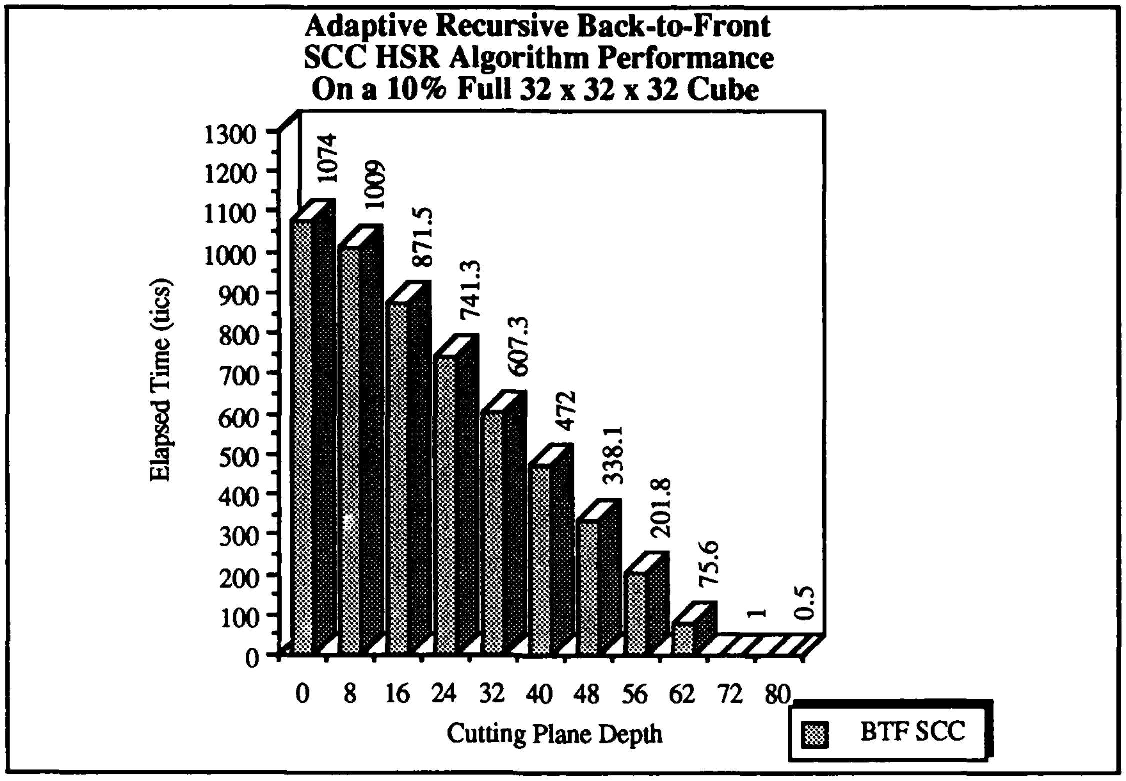


Figure 6.8: Performance of the Subcube-Center (SCC) Variant of the Adaptive Recursive Back-to-Front (BTF) HSR Algorithm with $\sigma = 10\%$.

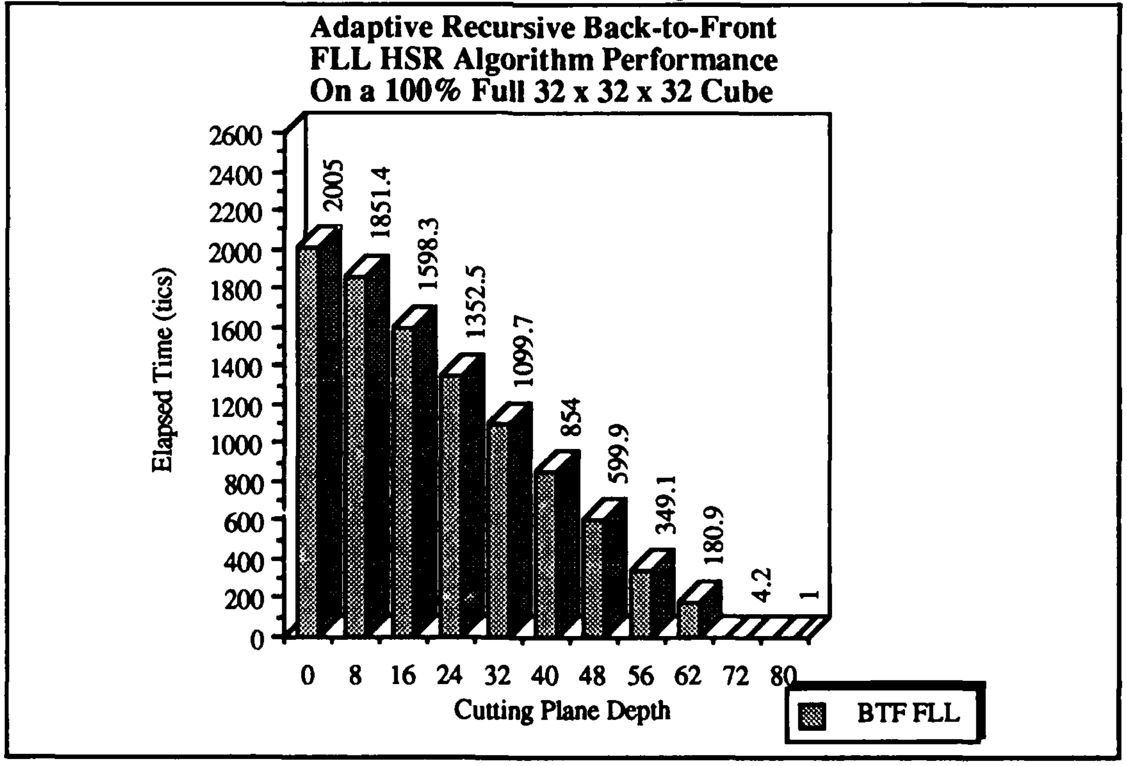


Figure 6.9: Performance of the Front-Lower-Left (FLL) Variant of the Adaptive Recursive Back-to-Front (BTF) HSR Algorithm with $\sigma = 100\%$.

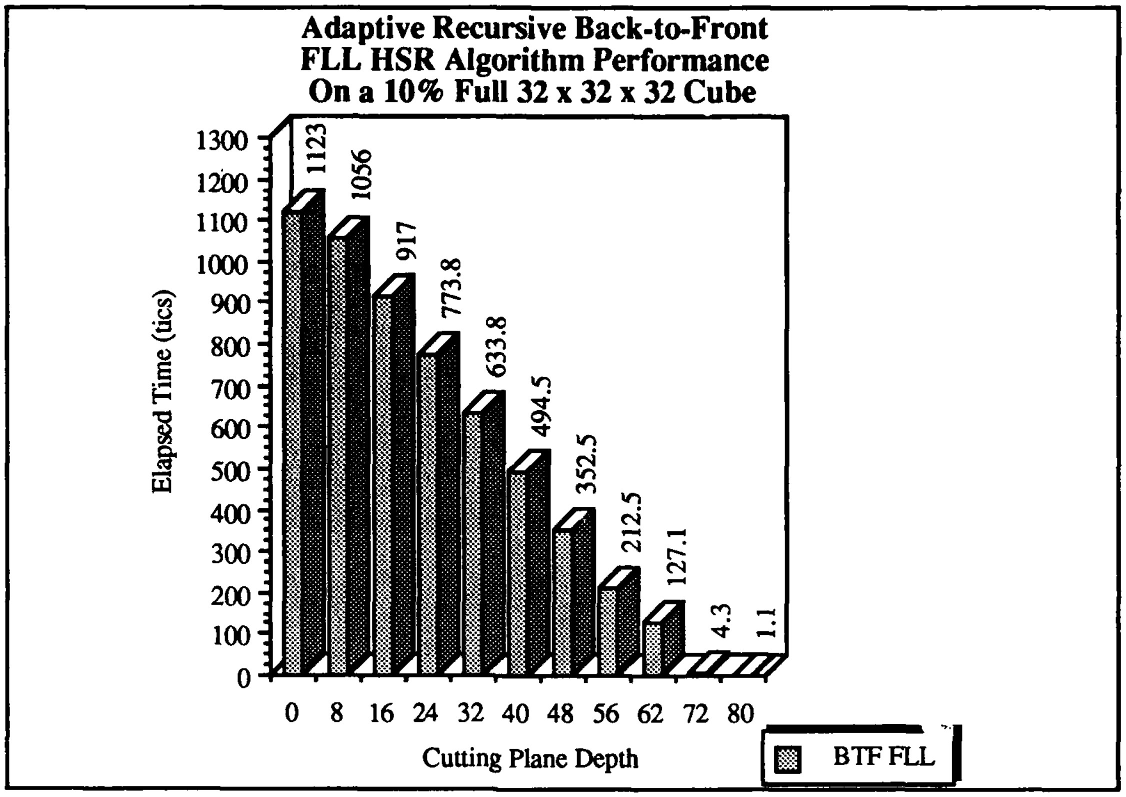


Figure 6.10: Performance of the Front-Lower-Left (FLL) Variant of the Adaptive Recursive Back-to-Front (BTF) HSR Algorithm with $\sigma = 10\%$.

saturation values. Testing results for both adaptive BTF HSR variants on 8^3, 16^3, and 32^3 voxel object space volumes are presented in Appendix I.

The timing results demonstrate that the performance of the adaptive BTF HSR algorithm does vary with cutting plane depth and σ, with the image rendering time of the algorithm declining with decreasing σ or as the cutting plane moves deeper into image space. On the average a noticeable performance improvement is attained. However, the worst case performance[1] for the algorithm is worse than the performance of the basic BTF HSR algorithm. This situation is not unexpected, because the adaptive BTF algorithm has a significant computational overhead due to the distance computations. One solution to this problem is to select either the basic or adaptive BTF HSR algorithm for use by each SSG

[1]Worst case performance is obtained when σ is large and the cutting plane is at the front of the image space volume.

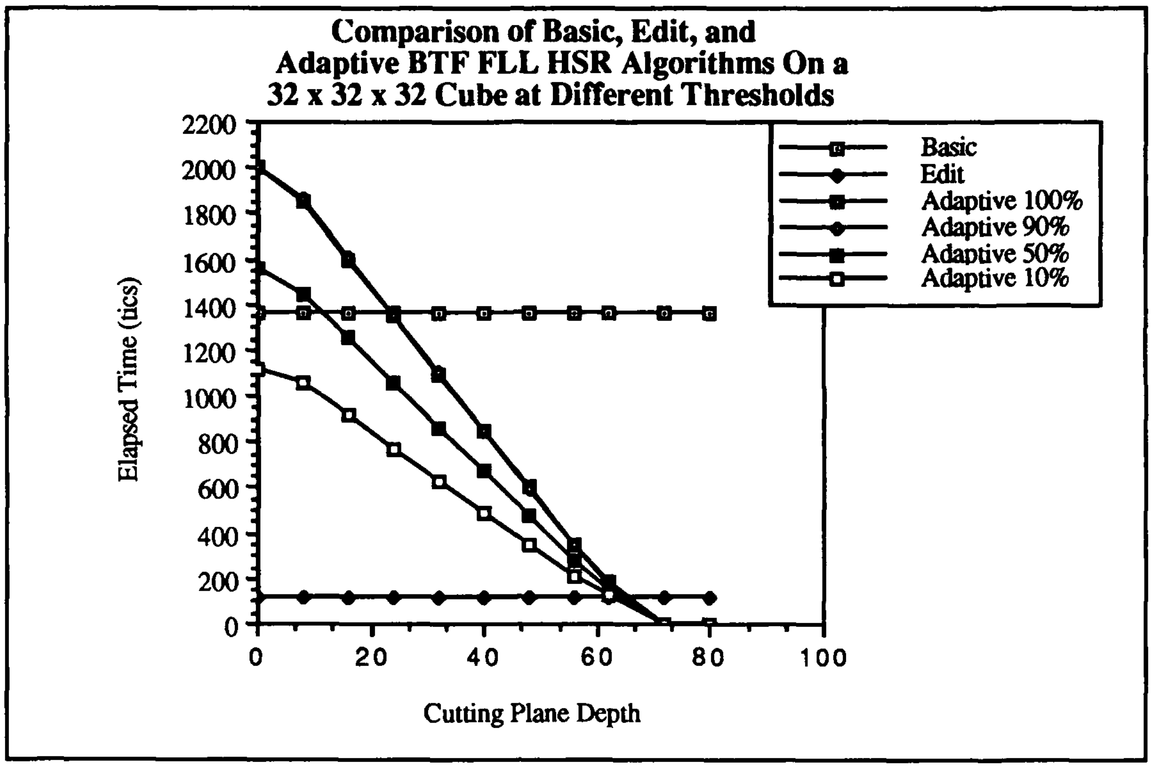


Figure 6.11: Comparison of the Elapsed Rendering Time Performance of the Adaptive, Scene Editing, and Basic Recursive Back-to-Front (BTF) Front-Lower-Left (FLL) HSR Algorithms.

based on cutting plane location relative to the SSG's image space volume. While this approach would work, I decided to take a different tack. The results clearly indicate that the adaptive termination approach can achieve significant performance improvement , so I decided to investigate employing the adaptive algorithm-termination strategy within the front-to-back algorithm. This new algorithm, and its performance, are discussed next.

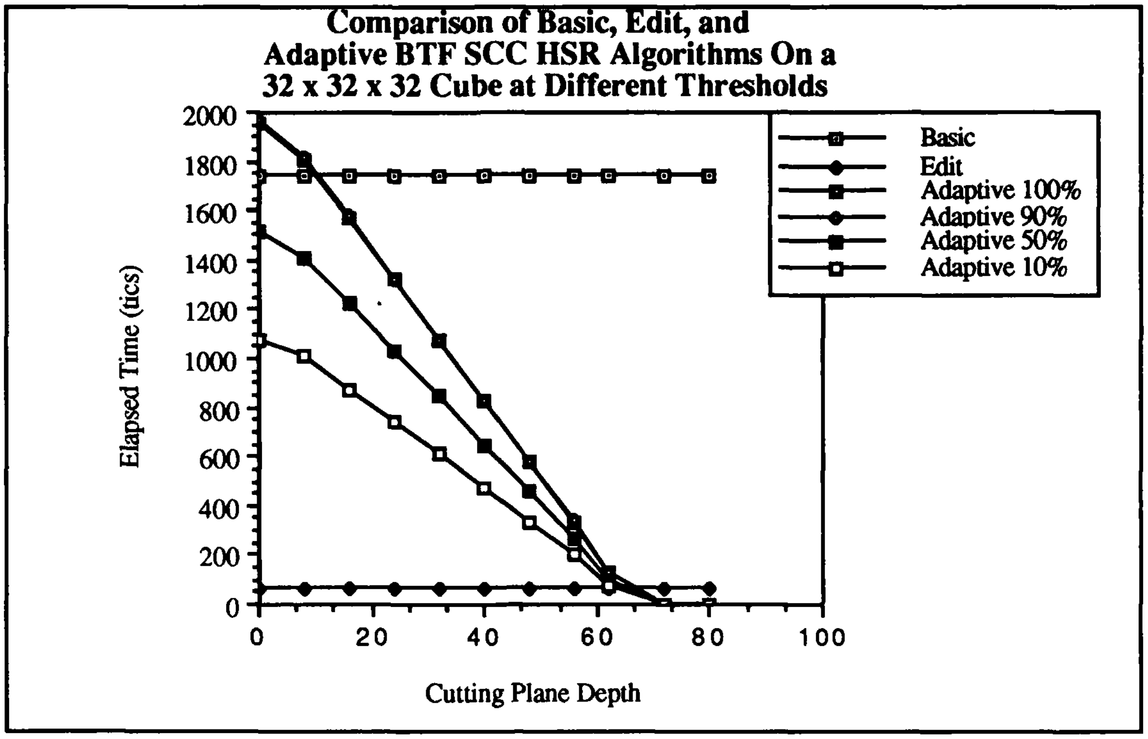


Figure 6.12: Comparison of the Elapsed Rendering Time Performance of the Adaptive, Scene Editing, and Basic Recursive Back-to-Front (BTF) Subcube-Centered (SCC) HSR Algorithms.

6.4 The Adaptive Recursive Front-to-Back Hidden-Surface Removal Algorithm.

The key difference between a BTF and FTB algorithm is that the FTB algorithm must check each image space array location for a value before writing to the location. As shown in Chapter V, this processing overhead makes the recursive FTB algorithm somewhat slower than its counterpart BTF algorithm, which implies, at first glance, that an adaptive FTB algorithm would not offer a performance improvement over the BTF algorithm. However, because the FTB algorithm processes from the front of image space, it generates information concerning the location of the screen space coordinates remaining to be written. This information can be used to direct its image rendering efforts and to adaptively terminate processing. Given the promising results obtained for the adaptive recursive BTF algorithm, I decided to develop an adaptive FTB algorithm. The critical change required to formulate the adaptive algorithm is employing a record keeping mechanism which guides image processing operations and detects the processing termination point.

An adaptively terminating FTB algorithm can stop processing when its output buffer is full, therefore a suitable record keeping mechanism is a data structure which keeps track of unwritten points in screen space. A previous, non-recursive, front-to-back approach to accelerating image production using this concept is described in [Rey87]. Reynolds' approach uses a linked-list data structure, called the dynamic screen, to maintain a record of the pixels which have been lit. The algorithm requires a simplification of the gray-scale image to an auxiliary image composed of binary-coded contours extracted from the object of interest at the desired viewing angle. A description of this technique is given in section 3.10.7. To achieve the same capability for adaptive algorithm termination using the original gray-scale data in a recursive HSR algorithm, a different processing strategy and record-keeping data structure must be employed.

The adaptive front-to-back hidden-surface removal algorithm described below offers two advantages over the dynamic screen algorithm. First, as presented in [Rey87], the dynamic screen FTB algorithm requires the formation of binary-contour images because it uses binary line segments and not gray-scale valued voxels to represent the scene. This approach, because of its use of thresholding or some other segmentation technique to cull the desired object from the volume, leads to object representation errors caused by the partial volume effect. The adaptive FTB HSR algorithm does not require the formation of contours, so the pre-processing overhead and representational errors which result from contour extraction are avoided. A second advantage is that the parallel processing implementation is simplified if the FTB and BTF algorithms employ a recursive HSR strategy.

The adaptive recursive front-to-back hidden-surface removal algorithm operates on an N x N x N[1] voxel object space and a 2N x 2N x 2N voxel image space to allow for supersampling. The vector sets $\{\mathcal{V}_O\}$ and $\{\mathcal{V}_I\}$, the generation points $\{\mathcal{P}_O\}$ and $\{\mathcal{P}_I\}$, and $\mathcal{R}$ are defined identically to those in the basic recursive FTB algorithm. As in the basic recursive FTB algorithm, the image space z-value for each image space point is generated and can be stored along with the density value obtained from the object space array for later use by shading hardware/software.

At this point the recursive HSR algorithm definition proposed in Chapter V provides some insight into a formulation for the algorithm. The definition states that the transformation of coordinates from object space to image space must be done in parallel, but note that the computations required to generate the coordinate values in each space need not be done simultaneously. Second, the definition does not address the image-space-to-screen-space transformation. This apparent omission is deliberate, because the generation of screen space coordinates from image space coordinates is a problem which is entirely separate from the object space to image space transformation process.

Consider the image space generation process from a new perspective as a three step process. The first step generates the image space voxel coordinates of each voxel in object space. The second step places the object space voxel density value into the image space voxel value, thereby gradually populating image space with voxel values. The third step, begun once image space is populated by transformed voxels, performs the image space to screen space transformation. This last transformation consists of a voxel-by-voxel search for the image space voxel with the smallest z' value meeting the user threshold criteria for each screen space coordinate. In a recursive algorithm, the image-space-to-screen-space transformation is extremely simple because screen space u and v coordinates correspond to

[1]$N=2^m$.

image space x' and y' coordinates. This correspondence allows the image space search to be finessed by locating the first[1] object space voxel meeting the user's selection criteria which is transformed to a given image space x', y' coordinate and then placing the value of the voxel at the matching screen space u,v coordinate. In this manner, the image is gradually built up. For the basic FTB algorithm, all object space voxels are transformed through image space into screen space, but only the first transformation to each pixel appears in the final image. The remaining computations are wasted. This perspective provides a guide to forming an adaptive termination recursive FTB algorithm by performing the entire image space coordinate generation step first, and then stopping processing when the image is complete in screen space. The crucial element is devising a method which allows the algorithm to determine when the image is complete. In broad terms, one strategy is to find the total number of the pixels which are lit when performing the image space coordinate generation step (since x',y' = u,v) and then to use this information to terminate processing when the total number of pixels lit during the image-space-to-screen-space transformation equals the total computed during image space coordinate generation.

First, assume that all of the voxels which will be transformed into image space in the next step meet the user's voxel density selection criteria. Then, generate the $\{\mathcal{N}_I\}$ which results from the FTB application of $\{\mathcal{R}\}$ to the $\{\mathcal{P}_I\}$ and $\{\mathcal{V}_I\}$ established by the user's scene orientation criteria. As each voxel coordinate in image space is generated, determine if the screen space pixel at the x',y' coordinates has been lit. If it is lit, do nothing. If it is not lit, increment the total number of screen pixels to be lit by one. When this step concludes, the total number of pixels required to be lit to render the image is in hand. Then, when performing recursive FTB HSR, stop processing when the total number of object space voxels transformed into image space and then written to screen

[1]Or in the case of a BTF algorithm, the last voxel to be transformed to image space from object space.

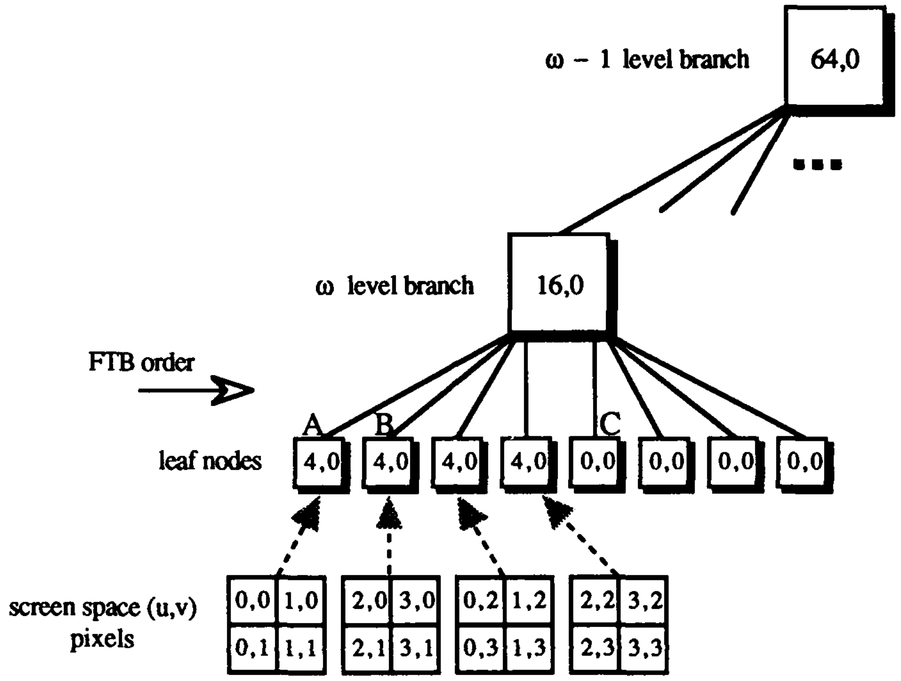
space equals the previously computed total number of pixels to be lit. Simply stated, the algorithm requires one $\{\mathcal{N}_I\}$ generation to determine the number of pixels to be lit, then a parallel $\{\mathcal{N}_I\}$ and $\{\mathcal{N}_O\}$ generation, which stops when the total number of pixels lit equals the total number required to be lit.

This basic sketch of the algorithm is unpractical because computations are repeated and because voids in screen space are filled in haphazardly. One way to increase the practicality of this approach is to keep a record of the initial image space coordinate generation process in a data structure, as the same sequence of computations is duplicated during the succeeding object space - image space transformation. A second modification is required to insure that an accurate 3D image is rendered at minimal computational cost. Recall that the original image space - screen space transformation step assumed that all the voxels would meet the user's criteria for selection. Since this is a patently false assumption when generating the image, holes occur in screen space, and deeper probes into image space are required to locate voxels which meet the criteria. Because of the manner in which recursive HSR algorithms move through object and image space, deeper probes can result in transforming large volumes of object space into image space in order to fill in one pixel. Rather than exploring a large image space volume for a suitable voxel, a controlled search of image space makes more sense, and since most of the computational cost is incurred at the lowest levels of recursion, the control should be most effective there. Therefore, a second required improvement is the use of a data structure for controlling the search through object space for voxels which transform successfully through image space into screen space.

To summarize, the data structure employed for adaptive termination must meet three criteria for efficient operation. First, the record keeping structure should maintain a record of which screen space pixels must be filled by the algorithm. Second, the structure must guide the search in image space, because the use of thresholding, and the normal operation of the imaging modalities, creates holes in screen space which must be filled in by object space voxels lying ever deeper in image space. Third, the data structure should capture the image space coordinate generation steps.

To guide the operation of the adaptive FTB algorithm, and to serve as a record keeping device, an oct-tree is used. This data structure was chosen because it mirrors the $\{\mathcal{N}_I\}$ decomposition in image space, therefore the coordinate values for the successively smaller partitions of $\{\mathcal{N}_I\}$ need only be computed when determining each suboctant's contribution to the final image. During image rendering, the branches of the oct-tree can be followed to retrace the image space processing steps already performed. An oct-tree, then, fulfills the requirement that the sequence of image space coordinates be computed once. The number of pixels predicted to be lit by a given suboctant as well as whether the suboctant has been transformed can be recorded within the corresponding oct-tree node. The octants which are predicted to have voxels which light screen space pixels can then be transformed first. If holes remain in the image, the remaining octants behind the transformed octant can then be operated upon. By retaining a pixel count and transformation status at each oct-tree node, the algorithm can move through object/image so that it does not transform octants lying in front of a cutting plane and only transforms octants behind the cutting plane to fill in holes in the image, thereby realizing the other two criteria. The operation of the adaptive FTB algorithm is sketched below.

First, an oct-tree is formed to retain the FTB image space coordinate generation sequence and the image space coordinates required to generate the individual image space voxel coordinate values. The initial operation of the algorithm requires one complete $\{\mathcal{N}_I\}$ generation to predict the number of pixels to be lit by each $\{\mathcal{N}_I\}$ of each oct-tree node and to record the image space coordinate generation sequence in the oct-tree. When all of the children of an image space branch have been processed, the total number of image space points predicted to be written by the corresponding octant for the branch is computed from the sums stored in its child nodes and stored in the branch node. To simplify the recursion, the total number of screen space coordinates to be generated by the oct-tree is copied from the root node of the oct-tree and placed in a global variable. The diagram in Figure 6.13 illustrates this step.



Legend

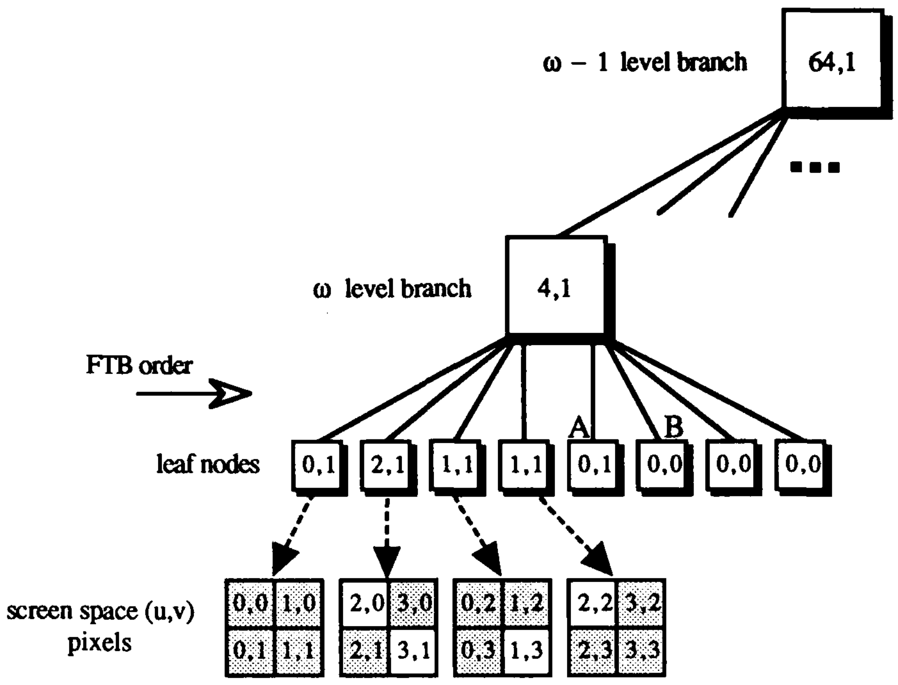
Data Structure	Contents
leaf or branch node	node total, examined status
screen space pixel	u-coordinate, v-coordinate

Figure 6.13: Image Space Coordinate Generation with the Adaptive Front-to-Back (FTB) Algorithm.

Figure 6.13 shows the status of the eight leaves of an ω-level (lowest level) branch after the $\{\mathcal{N}_J\}$ for each leaf has been processed by the adaptive FTB algorithm. The processing for the leaves has two parts. The first part calculates the coordinates of each leaf and retains these values within the node (not pictured) for later use in rendering the image. The second portion of the processing determines the screen space pixels which each leaf is predicted to light, and this number is stored within the leaf as the node total. For example, in Figure 6.13, the leaves marked A and B are both predicted to light four pixels, and the leaf marked C is predicted to light none. In addition, each leaf has been marked as unexamined (this flag is used later in the image rendering phase) by setting the examined status flag for each leaf to zero. Since image space is processed in FTB order, only the leaf octants lying at the front of image space are predicted to light any pixels during image rendering, rear octants show a node total of zero indicating that they are completely obscured and can not contribute information to the final image unless holes remain after the foremost octants are processed. The processing at the branches of the oct-tree consists of maintaining a running total of the number of pixels predicted to be lit by all its leaves. So, again referring to Figure 6.13, the ω-level branch has a node total of 16, indicating that its child leaves are predicted to light 16 pixels, and the ω-1 level branch has a node total of 64, indicating that the leaves from its child branches are predicted to light 64 pixels.

Once the initial $\{\mathcal{N}_I\}$ generation is complete, 3D image rendering is performed by recursively moving through $\{\mathcal{N}_I\}$ and $\{\mathcal{N}_O\}$, using the oct-tree to generate the image space coordinates in $\{\mathcal{N}_I\}$. If an image space octant or suboctant is totally obscured it has a node total value of zero, indicating that its associated object space octant need not be transformed into image space when rendering the image. The initial pass over the leaves of the lowest-level (ω) branch only examines leaves with a total greater than zero. Whenever an object space point is transformed into image space and thence to screen space, the total at the oct-tree node is decremented by one, the total at the leaf's parent is decremented by one, and the global total value is decremented by one. As soon as all eight voxels at a leaf have been processed, leaf processing stops and the next leaf in FTB order is processed. Whenever an image space oct-tree leaf is tested for transformation, this fact is recorded by setting the examined status flag to one so that the node is never tested nor transformed again.

After the leaves for the ω-level branch with an initial node total > zero are all transformed, the image is tested for holes. At the lowest level, say ω, in the image space.



Legend

Data Structure	Contents
leaf or branch node	node total, examined status
screen space pixel	u-coordinate, v-coordinate

Figure 6.14: Scene Rendering with the Adaptive Front-to-Back (FTB) Algorithm.

oct-tree, the existence of holes is demonstrated by the ω level branch having a node total value > 0 when the last of its leaves which had an initial node total value > zero have been transformed. If a hole remains, then octants corresponding to the un-projected oct-tree leaves lying further back in image space are projected into screen space through image space from object space. When the total at the ω level branch equals zero, the object-space-to-image-space transformation process for this node stops, and the algorithm proceeds on to process the leaf nodes of another ω level branch. This phase of the operation of the algorithm is depicted in Figure 6.14, which shows the oct-tree from Figure 6.13 at a point during the image rendering operation.

Figure 6.14 depicts the processing status after the first five leaves of the ω-level branch have been examined. The first leaf lit all four of the pixels it was predicted to light, so its total is zero and its examination status is 1, indicating that the corresponding image space volume for the node was examined. The second leaf only lit two pixels, so its total is two, but its examination status is also 1. The other two leaves predicted to light pixels were also not completely successful, as indicated by their totals. Since the ω-level branch total is greater than zero after the initial pass over all the leaves expected to light pixels is completed, the remaining leaves belonging to the branch are examined in FTB order. In the diagram, node A has already been examined, but since the ω-level branch's total is unchanged the leaf lit no pixels. The leaf at B is the next leaf to be examined.

Because of the spatial relationships between the leaf nodes belonging to higher level branches, the octant transformation process cannot be terminated when the total at an upper-level branch drops to zero. Instead, the algorithm must take on a broader examination for pixels which can be lit, and the entire image space octant belonging to the branch must be re-examined. Termination of octant re-examination is permitted only when the global total number of pixels remaining to be lit reaches zero. This requirement, in effect, calls for a complete search of object space for all suitable voxels which transform into the image space octant and can be mapped into any pixel in screen space. At this point, the record keeping concerning the leaves and branches which were already inspected becomes important. The examination status records allow the octant-wide search to concentrate on the octants in image space which were originally by-passed since only by-passed octants can contribute to the final image. Because only the suboctants which can contribute to the final image are projected during this broader search, and as over 80% of the computation occurs in the leaves and lowest level branches, a performance gain is realized.

In order to enhance the operation of the adaptive FTB algorithm, the object space pre-generation optimization, as described in the adaptive BTF algorithm, is performed.

The use of oct-trees to hold both object space and image space is feasible for the adaptive FTB algorithm for the same reasons as object space pre-generation is effective for the adaptive BTF algorithm. These are: 1) the distributed memory architecture allows the oct-tree to remain within memory at all times and 2) the partitioning of object space among the SSGs ensures a low oct-tree memory requirement at the SSGs. Pseudocode for the complete algorithm including the the object space pre-generation optimization is contained in Appendix J.

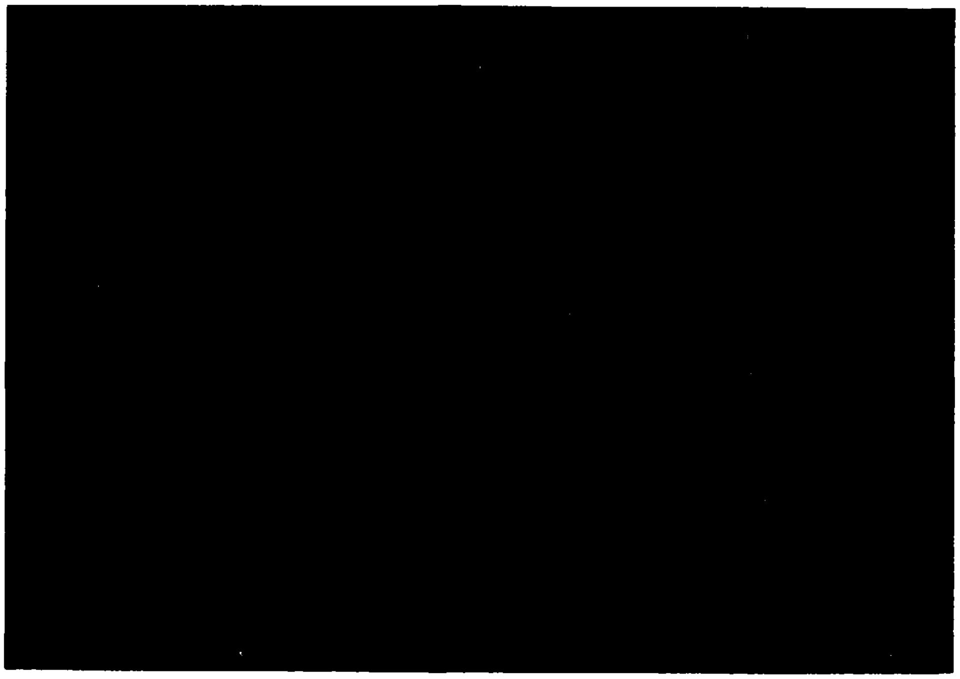


Figure 6.15: Human Vertebrate in Gel Rendered in the Hypercube Interconnection Topology Using the Adaptive Front-to-Back (FTB) Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 0° z-axis Rotation and $\sigma = 100\%$.

The image rendering accuracy assessment methodology described in section 5.2.4 demonstrates that the adaptive BTF HSR algorithm generates correct images. Using the adaptive recursive FTB SCC HSR algorithm, a 64^3 voxel volume depicting a human vertebrate in gel was rendered using the hypercube interconnection topology with 64k byte

Figure 6.16: Mitre Box Rendered in the Hypercube Interconnection Topology Using the Adaptive Front-to-Back (FTB) Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 0° z-axis Rotation and $\sigma = 100\%$.

message packets at 0° rotation z-axis rotation. The resulting image is presented in Figure 6.15.

Using the Adaptive Recursive Front-to-Back (FTB) SCC HSR algorithm, a 64^3 voxel volume depicting a mitre box was rendered using the hypercube interconnection topology with 64k byte message packets at 0° rotation as well as 45° x-axis and 45° y-axis rotation. The resulting images are presented in Figures 6.16 and 6.17.

Using the timing methodology described in section 5.2.4, the performance of the adaptive FTB HSR algorithm was determined. As demonstrated in Chapter V, the performance of the pipeline can be approximated by the performance of the SSGs, so the discussion is limited to the performance of the adaptive FTB HSR algorithm operating

Figure 6.17: Mitre Box Rendered in the Hypercube Interconnection Topology Using the Adaptive Front-to-Back (FTB) Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 45° x-axis and 45° y-axis Rotation and $\sigma = 100\%$.

within each SSG. To determine if the amount of time required to perform HSR in the SSG stage changes with the depth of the cutting plane or rotation, the adaptive recursive FTB HSR algorithm was timed for a range of cutting plane depths, rotations, threshold window saturation values, and scene sizes. These results are presented graphically in Figures 6.18 through 6.21 for both the SCC and FLL variants of the algorithm on 32^3 object space voxel volumes for $\sigma = 10\%$ and 100%. Complete results for both variants for 8^3, 16^3, and 32^3 scene sizes across a range of densities are contained in Appendix K.

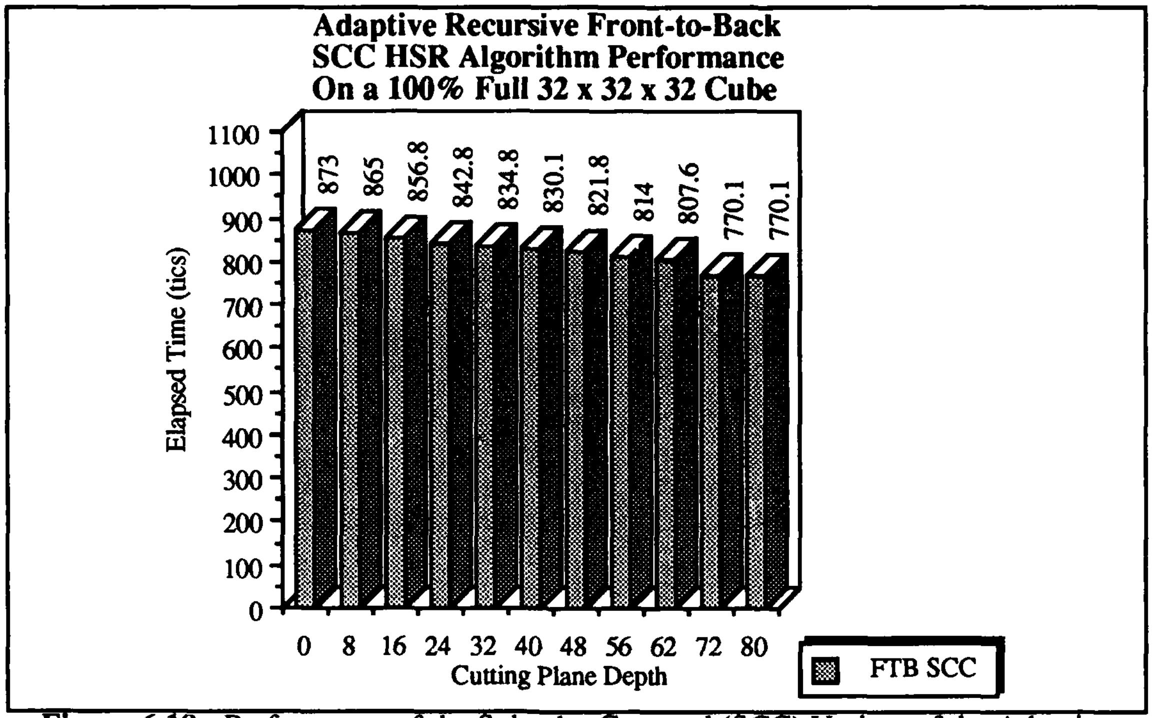


Figure 6.18: Performance of the Subcube-Centered (SCC) Variant of the Adaptive Recursive Recursive Front-to-Back (FTB) HSR Algorithm with σ = 100%.

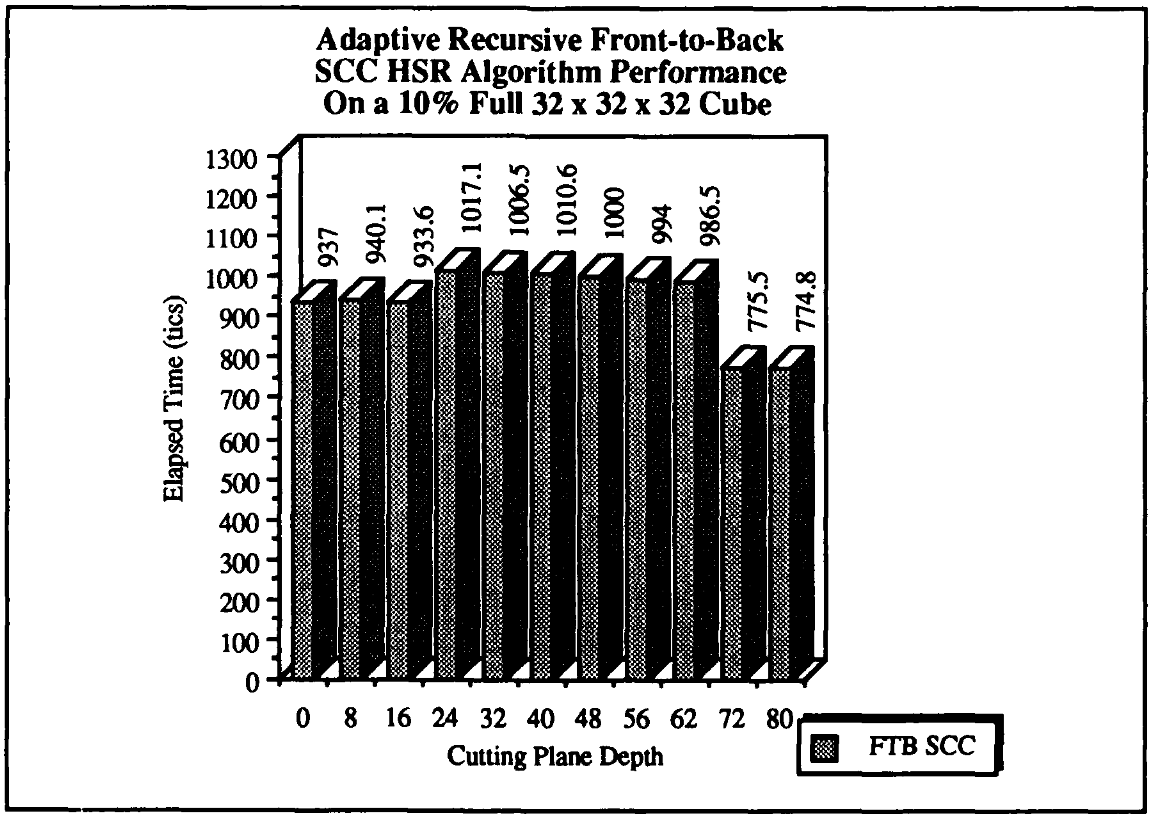


Figure 6.19: Performance of the Subcube-Centered (SCC) Variant of the Adaptive Recursive Front-to-Back (FTB) HSR Algorithm with σ = 10%.

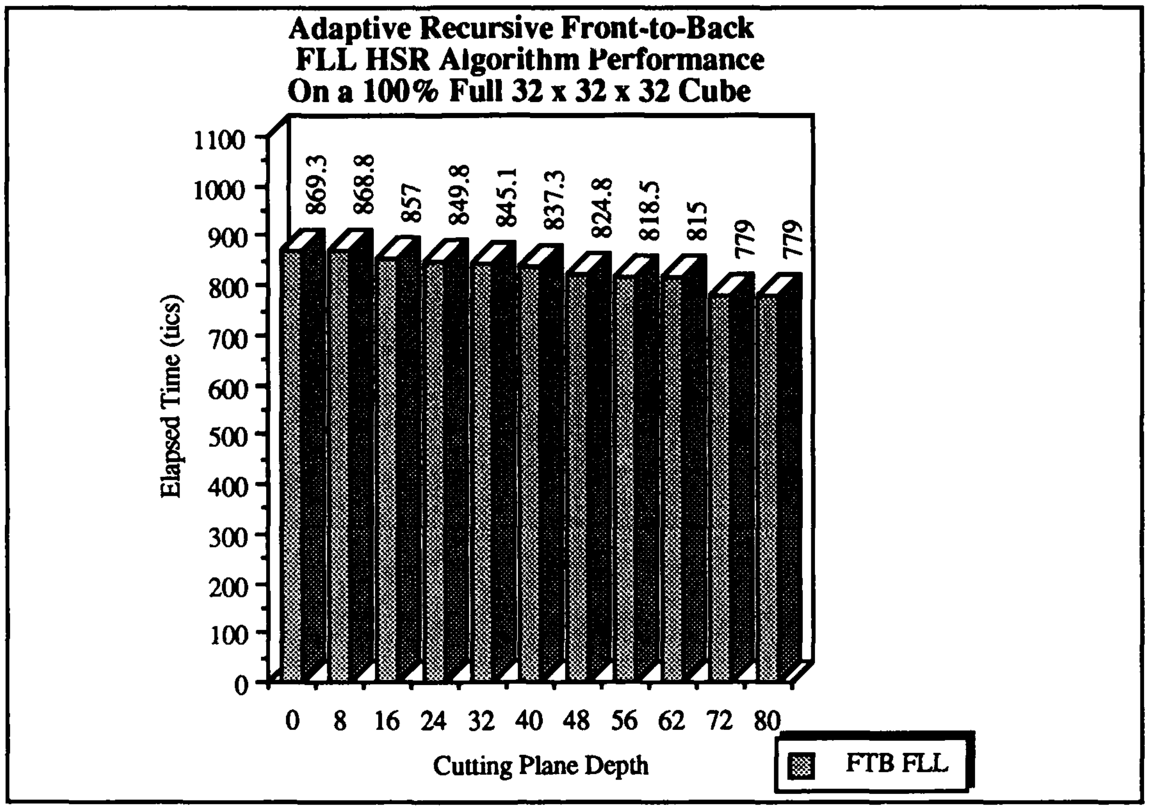


Figure 6.20: Performance of the Front-Lower-Left (FLL) Variant of the Adaptive Recursive Front-to-Back (FTB) HSR Algorithm with σ = 100%.

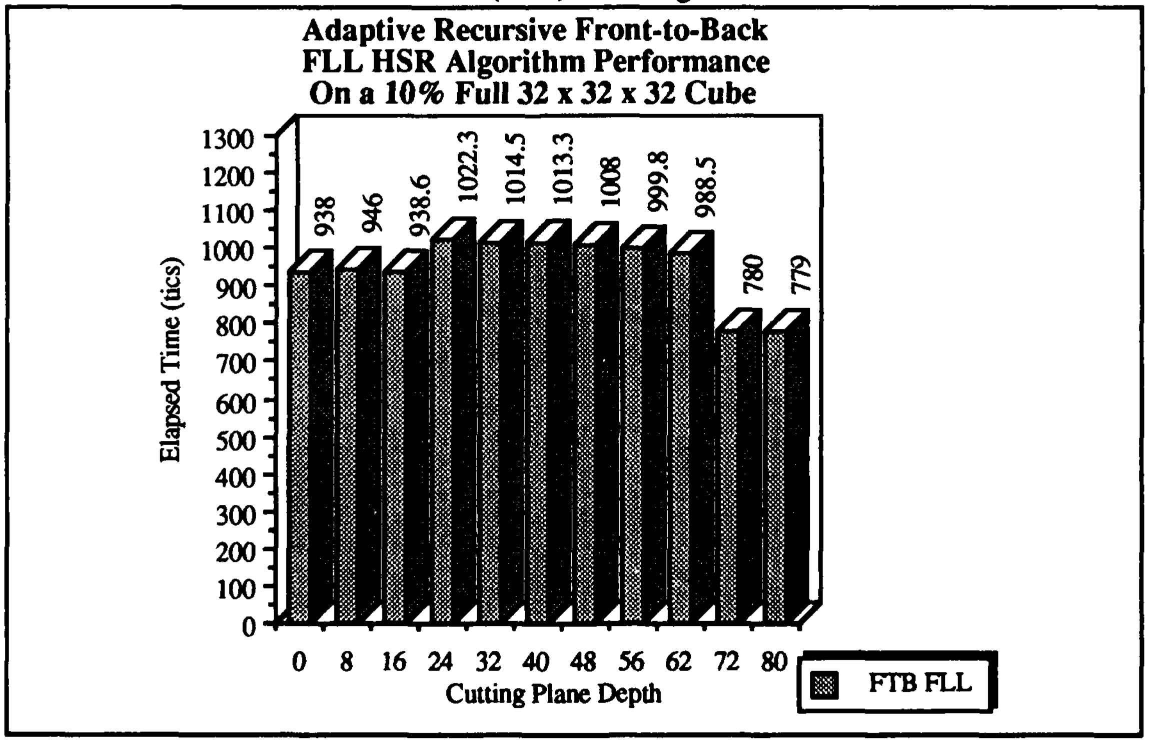


Figure 6.21: Performance of the Front-Lower-Left (FLL) Variant of the Adaptive Recursive Front-to-Back (FTB) HSR Algorithm with σ = 10%.

A comparison of the performance of the FLL and SCC variants of the basic, scene editing, and adaptive FTB HSR algorithms with no rotation is presented in Figures 6.22 and 6.23 for a scene size of 32^3 voxels in object space across a range of threshold window saturation values. Complete results for both variants on 8^3, 16^3, and 32^3 scene sizes are contained in Appendix K.

The performance of the adaptive FTB algorithm, as indicated by the foregoing performance graphs, is as expected. The image rendering time for a given σ is fairly consistent for different cutting plane locations due to two capabilities in the algorithm. First, the algorithm can omit processing octants at the front of the image and begin processing where the cutting plane intersects an octant. This capacity, combined with the

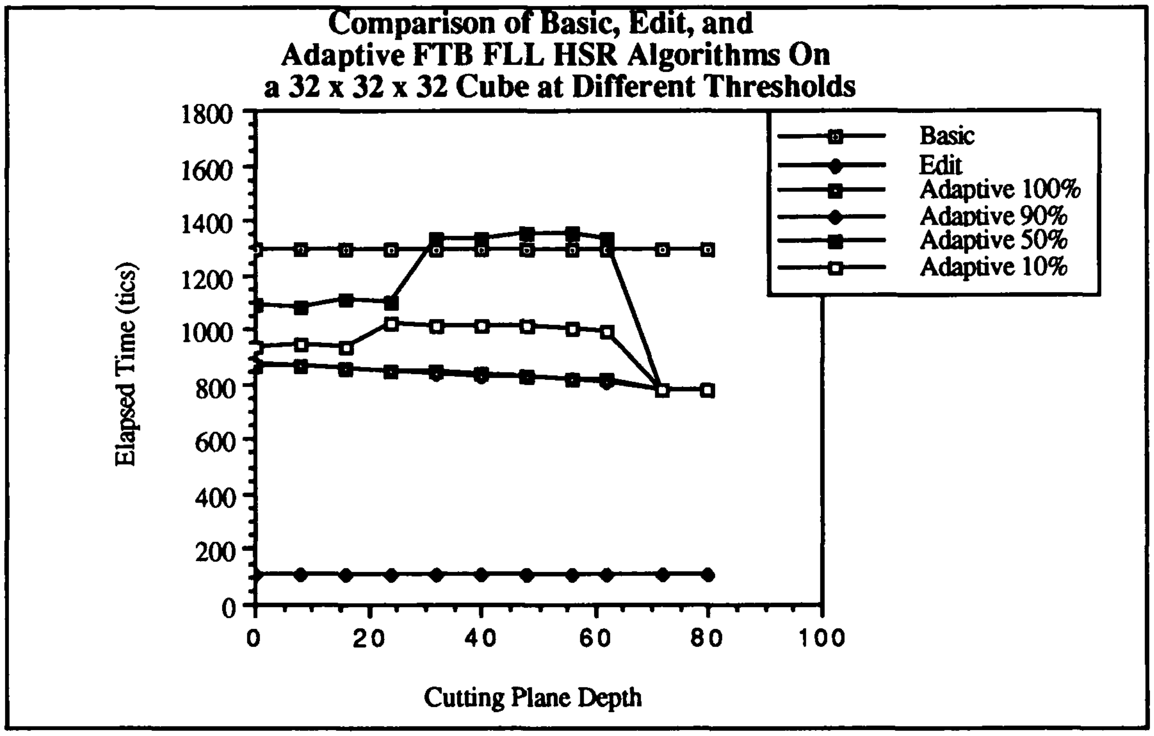


Figure 6.22: Comparison of the Elapsed Rendering Time Performance of the Adaptive, Scene Editing, and Basic Recursive Front-to-Back (FTB) Front-Lower-Left (FLL) HSR Algorithms.

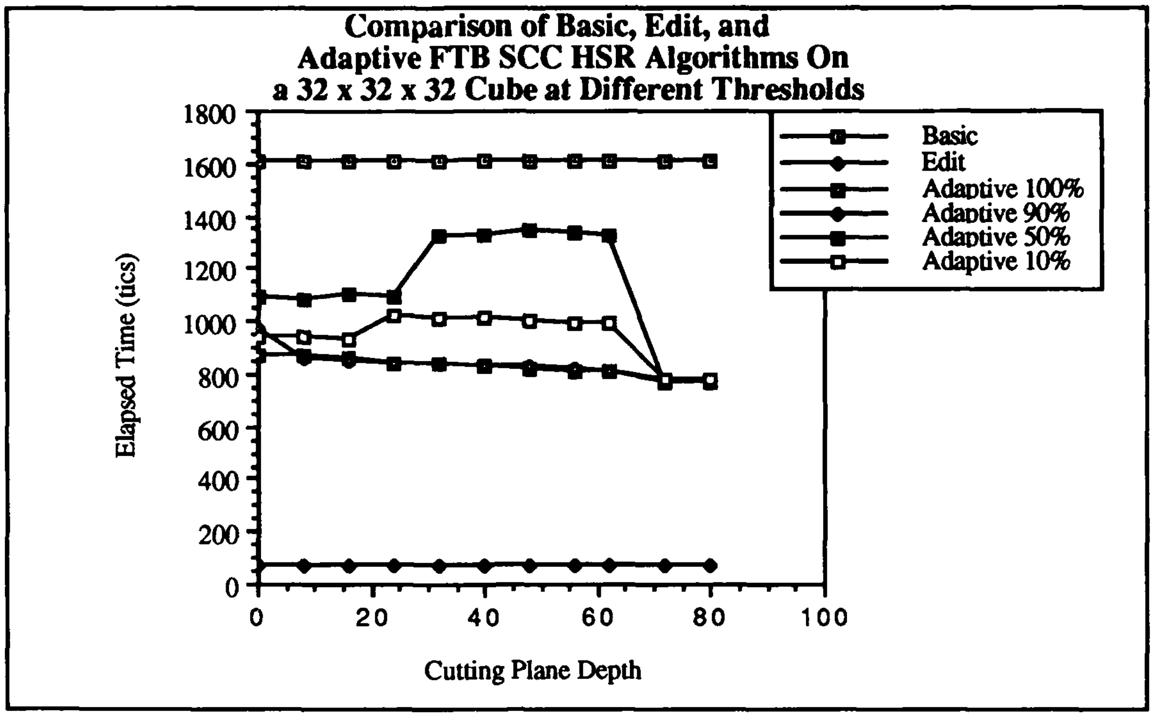


Figure 6.23: Comparison of the Elapsed Rendering Time Performance of the Adaptive, Scene Editing, and Basic Recursive Front-to-Back (FTB) Subcube-Centered (SCC) HSR Algorithms.

capability for image rendering processing termination when the image is complete, insures consistent performance for a given σ.

On the other hand, a change in σ affects the performance of the algorithm noticeably. As σ deceases, the image rendering time increases until σ = 50%, then the image rendering time decreases, but not to the level achieved when σ = 100%. This performance is explained by the interaction of the object space oct-tree and the image space oct-tree. For large σ values, many of the image space voxels predicted to be written to screen space are filled by object space voxel values which are accepted by the threshold window. The high acceptance rate precludes the need to engage the mechanisms which search through image space for additional voxels with which to fill in screen space holes. For low σ values, many of the image space voxel values predicted to fill in screen space are filled by object space voxel values which fail the distance test. The existence of holes forces continued transformation of object space voxels into image space. This processing results in many of the image space oct-tree ω-level branches being reached in the search for voxels having density values within the threshold window. However, the object space oct-tree prevents examination of the leaves of the image space oct-tree because the density value is known before the image space leaves are processed. Since most of the image rendering cost is incurred when processing the image space oct-tree leaves, eliminating these fruitless computations while trying to fill-in screen space holes reduces elapsed image rendering time. The worst possible processing situation occurs when a large proportion of voxels values in the object space oct-tree fall within the threshold window, but the number of holes to fill are relatively few. This processing situation results in many of the object-space-to-image-space transformation computations being wasted because a large number of screen space positions are already occupied by a density value. This situation occurs at σ = 50%.

With the intent of providing more consistent image rendering times and a higher image rendering rate than is possible with either the adaptive FTB or BTF algorithms, I decided to investigate the performance which can be achieved by dynamically selecting the algorithm with superior predicted performance for a given processing situation. The results of this inquiry are presented in the next section.

6.6 Dynamic Adaptive Hidden-Surface Removal.

An enhancement to the performance of the SSG stage can be attained by an SSG-specific selection of either the adaptive BTF or adaptive FTB HSR algorithm based on the image rendering situation within each SSG. The selection process takes advantage of the consistently low image rendering times achieved with the adaptive FTB algorithm when high σ values are realized within an SSG or when the cutting plane is near the front of the portion of image space processed by the SSG. In addition, the algorithm selection procedure capitalizes on the consistently low image rendering time achieved by the adaptive

BTF algorithm for low σ values within an SSG or when the cutting plane is near the back of the portion of image space processed by the SSG.

This new scene-processing strategy improves SSG performance by having each SSG in the multiprocessor image rendering machine independently and dynamically select, for each user input, the best performing adaptive recursive HSR algorithm. Algorithm selection is based on the location of the cutting plane within the $\{\mathcal{N}_I\}$ for each processor and the threshold window saturation[1] (fullness) within the volume to be displayed. In general, a cutting plane close to the front of the scene indicates that a recursive front-to-back algorithm should be used, while a plane location at the back calls for the use of recursive back-to-front algorithm. Threshold window saturation affects the cutover depth by advancing the cutover point as threshold window saturation decreases. The remainder of this section describes an approach to selection of the algorithm cutover point and resulting SSG performance.

The basis for selection of either the adaptive FTB or the BTF algorithm is the z'-dimension location of the cutting plane in image space and σ_e. This strategy is justified by two observations. First, for cutting planes close to the front of the scene, the adaptive BTF algorithm must process most of the data, whereas the adaptive FTB algorithm processes relatively little, giving the FTB algorithm a performance edge. As the cutting plane moves deeper into the scene, the two algorithms approach the same performance until, at the cutover point, the overhead of the adaptive FTB algorithm equals the pixel overwriting cost of the adaptive BTF algorithm. From the cutover point on to the back of the scene, the adaptive BTF algorithm is faster than the adaptive FTB algorithm. These ideas are

[1]The threshold window saturation, σ, is the percentage of the voxels within the $\{\mathcal{N}_0\}$ of an SSG which fall within the threshold window set by the user. Because the threshold window saturation is unknown before the image is formed, it is approximated by the expected threshold window saturation, σ_e. The expected value is obtained by histogramming $\{\mathcal{N}_0\}$ as each SSG loads its data set and then computing each density value's percentage population within $\{\mathcal{N}_0\}$. The value for σ_e is obtained by summing the density value percentage population values of the density values which lie within the threshold window.

illustrated in Figure 6.24 where a cutting plane dividing a volume in image space is depicted. For illustrative purposes, the volume is divided between two processors. The cutting plane lies close to the front of the portion of image space operated upon by processor one, so this processor should employ the adaptive FTB algorithm to generate its image. However, the cutting plane lies close to the back of the volume processed by processor two, so the best choice for this processor is the adaptive BTF algorithm.

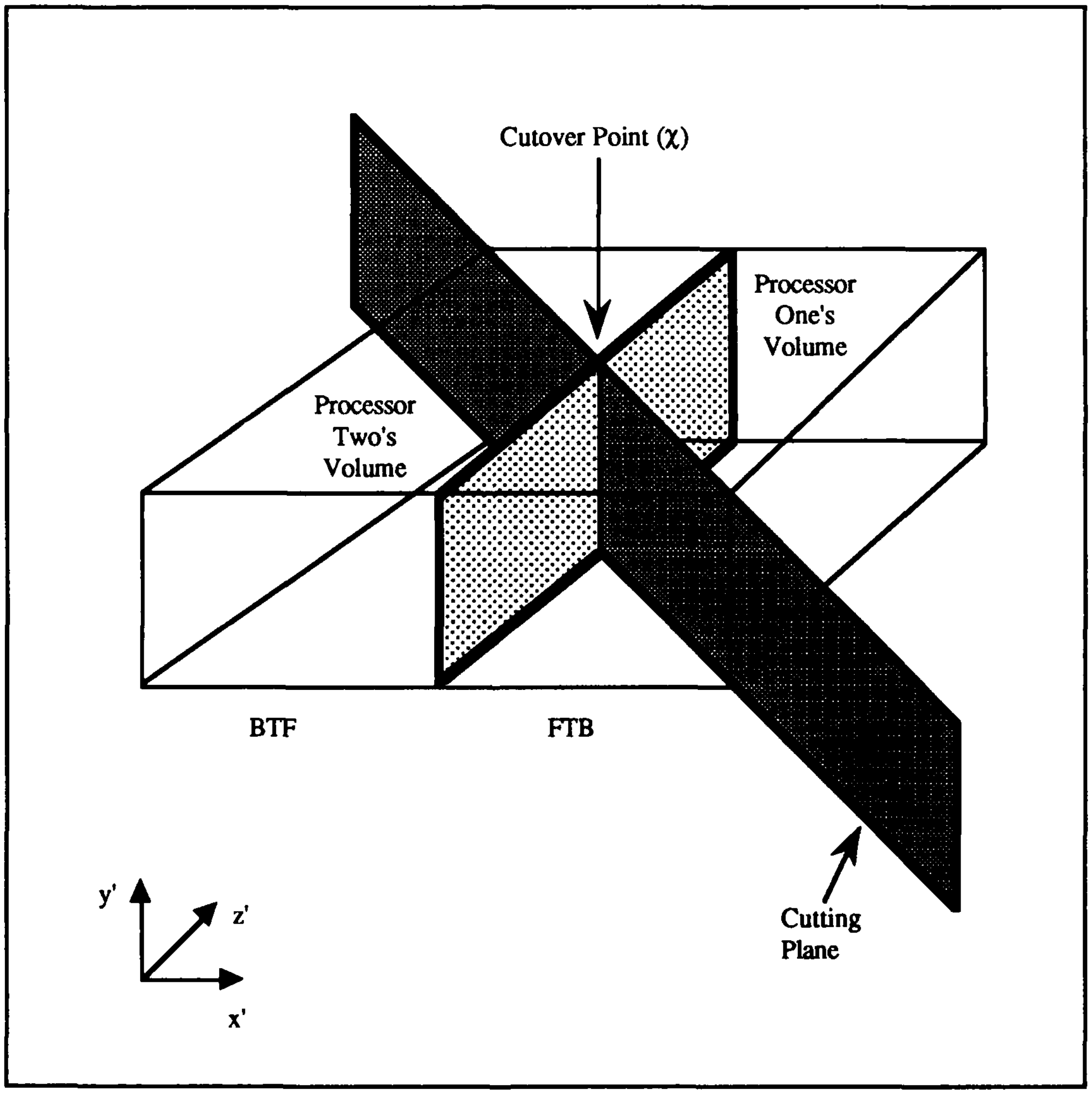


Figure 6.24: The Influence of Cutting Plane Location on Processor Selection of Either the Front-to-Back (FTB) or Back-to-Front (BTF) Hidden-Surface Removal Algorithm.

The second observation is that, for low σ_e values, the adaptive BTF algorithm computes relatively few image space coordinates because of its use of the pre-generated object space oct-tree. This is in contrast to the FTB algorithm which generates all of the image space coordinates once when pre-generating the image space oct-tree, and then generates a substantial fraction of these same coordinates again when attempting to fill in screen space with a sparsely populated image space.

Given this basic strategy for dynamic algorithm selection, I decided to investigate the positioning of the cutover point, χ, by gathering SSG image rendering elapsed time performance data on each of the four adaptive HSR algorithm variants for x-axis and y-axis rotation from 0° to 360°. This testing was combined with the examination of the effect of the threshold window saturation, σ, upon the performance of the algorithm by using σ values of 10%, 50%, 90%, and 100%. For each σ, the effect of voxel value distribution within $\{\mathcal{N}_O\}$ was determined for the extreme cases: desired voxel values are clumped in a compact sub-volume within $\{\mathcal{N}_O\}$ or desired voxel values are randomly distributed within $\{\mathcal{N}_O\}$. Results indicate that neither random nor clumped distribution affects adaptive BTF algorithm image rendering speed but that the adaptive FTB algorithm performs better for clumped than random distributions. Therefore, to encompass the expected performance range of the dynamic HSR algorithm, the following analysis is based upon timing results obtained from the random as well as the clumped voxel value distributions.

The initial round of testing demonstrated that the two FLL-based algorithm variants do not offer as much performance gain using this approach as do the SCC-based algorithms. Testing on the FLL-based variants demonstrated that for several different x- and y-axis rotations, the cutover point falls to the rear of $\{\mathcal{N}_I\}$. This performance is attributed to the location of P_O and P_I within $\{\mathcal{N}_O\}$ and $\{\mathcal{N}_I\}$ for the FLL approach. As was the case for the basic and edit BTF and FTB algorithms, the performance of the adaptive BTF FLL algorithm suffers because the cutting plane intersection test examines the volume around P_I for an intersection, with the volume defined by the distance from P_I to

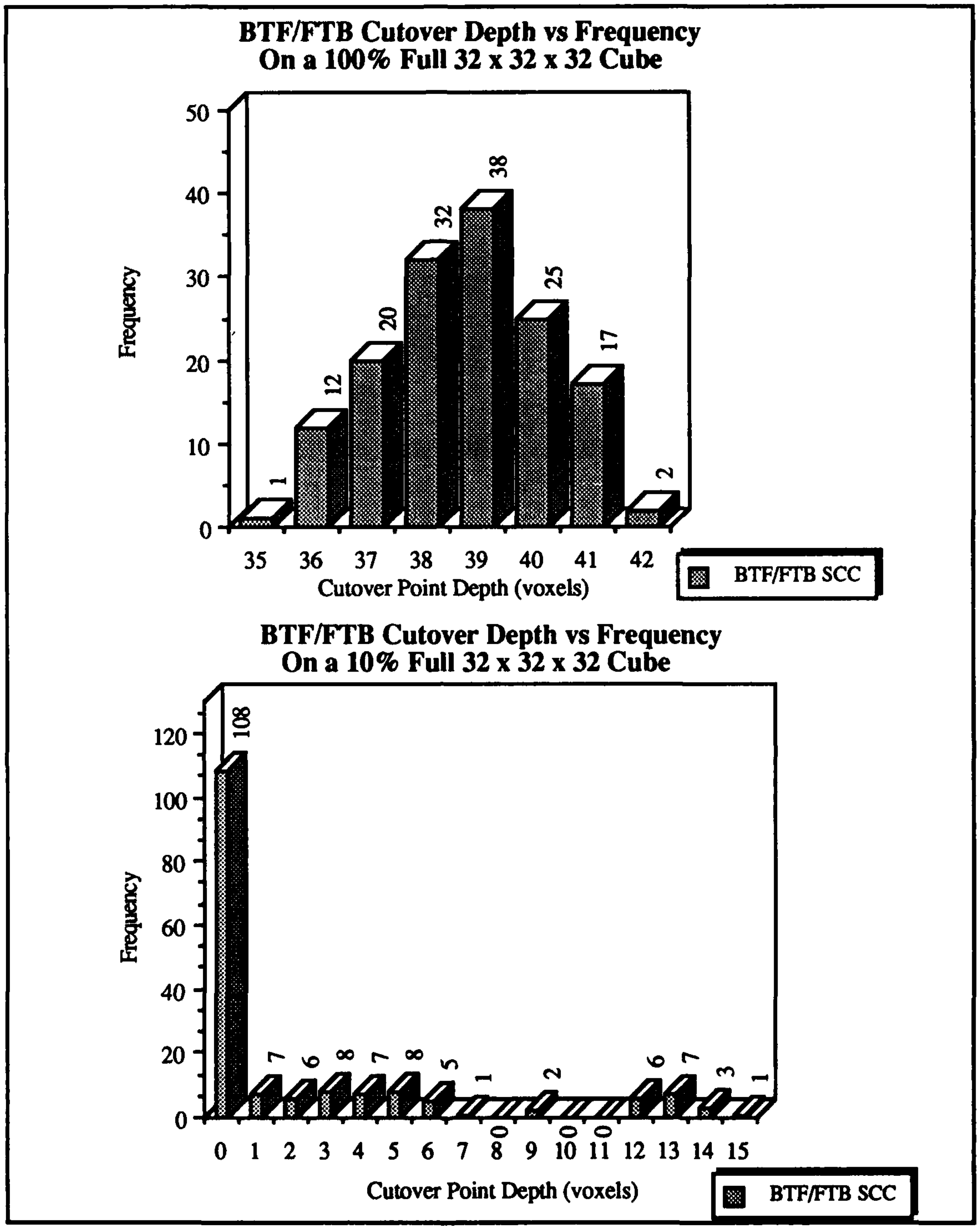


Figure 6.25: Algorithm Cutover Location Frequency Distribution for Threshold Window Saturation Values of 100% and 10% Using the Adaptive Front-to-Back (FTB) and Adaptive Back-to-Front (BTF) Subcube-Centered (SCC) HSR Algorithms.

the furthest point within $\{\mathcal{N}_I\}$. Because the test volume is eight times larger than $\{\mathcal{N}_I\}$, the algorithm spends additional time working in octants of $\{\mathcal{N}_I\}$ and $\{\mathcal{N}_o\}$ which do not

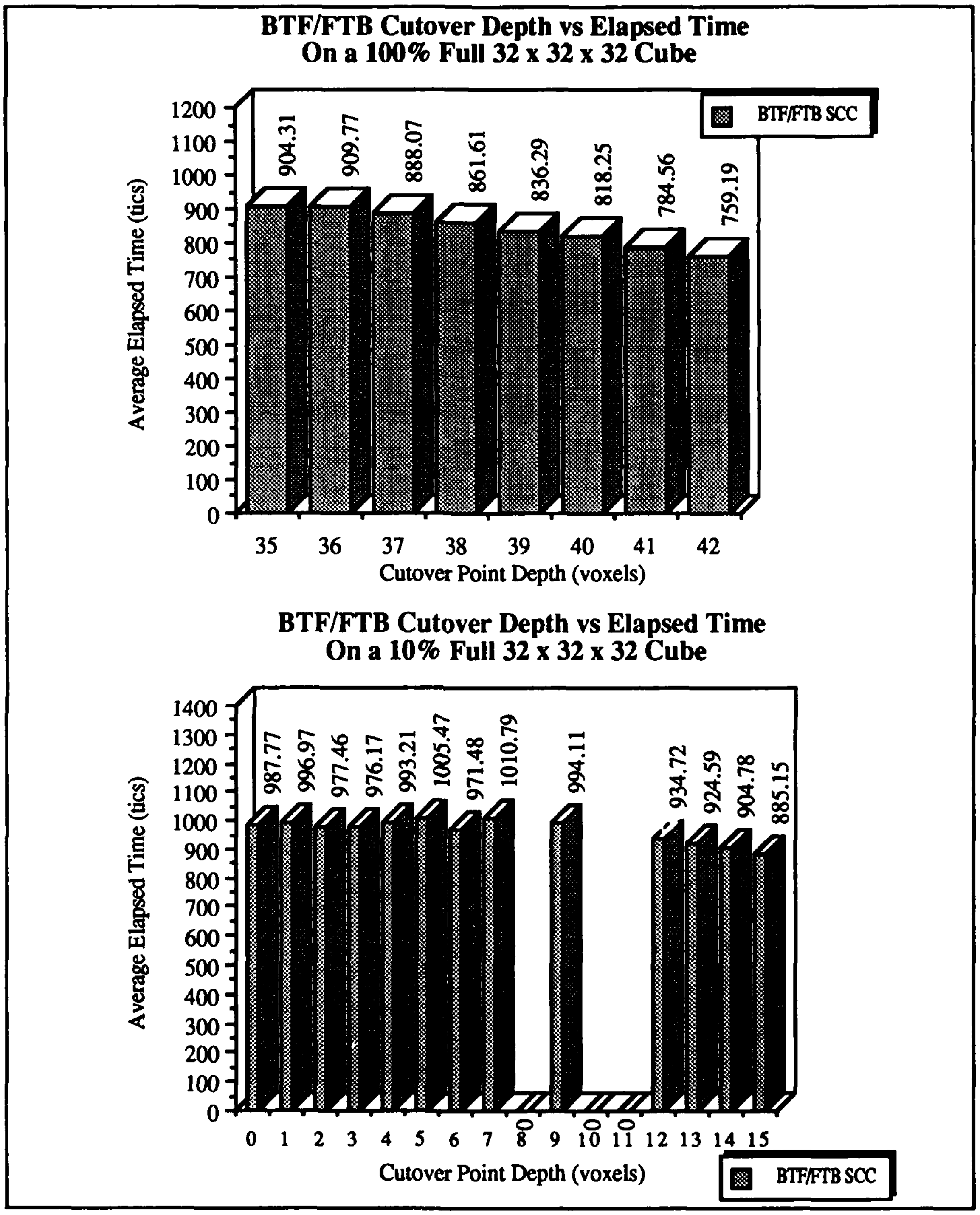


Figure 6.26: Average Elapsed Time at Algorithm Cutover Location for Threshold Window Saturation Values of 100% and 10% Using the Adaptive Front-to-Back (FTB) and Adaptive Back-to-Front (BTF) Subcube-Centered (SCC) HSR Algorithms.

appear in the final rendered image. The SCC-based algorithms do not have this performance characteristic, therefore I investigated the performance of these algorithms in depth. The SCC-based Dynamic Adaptive HSR algorithm and performance is described in the following paragraphs.

Upon completion of the SSC-based testing of the adaptive FTB and BTF algorithms, the data revealed that χ varies with the angular rotation about the x- and y-axes and with σ. The frequency distribution for χ within a 32^3 object space cube for $\sigma = 10\%$ and $\sigma = 100\%$ is presented in Figure 6.25. The complete set of frequency distributions obtained for the SCC-based algorithm for all four σ values and three object space cube sizes is contained in Appendix L.

The image rendering time at the cutover points was also computed, and is presented in Figure 6.26 for $\sigma = 10\%$ and 100%. The complete set of cutover point elapsed times for each cutover position for all four σ values and three object space cube sizes is contained in Appendix L.

From examination of this data and other data sets, it is apparent that χ moves forward within the object space cube for each SSG as the value of σ decreases. This result is expected because the performance of the adaptive BTF algorithm improves with decreasing σ, while the performance of the adaptive FTB algorithm initially worsens and then improves to nearly the $\sigma = 100\%$ level. Given that χ moves forward with decreasing σ, and does so smoothly, χ versus σ was plotted for the test rotation cases and object space $\{\mathcal{N}_o\}$, yielding a distribution of cutover locations within the volume. The plot indicated that there is a clear central tendency to the location of χ, but that there is also a wide distribution around this central point. As no single value was clearly superior, an analytic expression for the cutover point for a given χ is required. To prevent premature cutover of the algorithms, the largest χ value for each σ was selected from each χ vs σ plot, and these points were used to derive a second degree polynomial expression for cutover point location. This procedure yielded the following equations.

For a 8 x 8 x 8 object space cube: $\chi = 6.25\sigma^2 + 1.25\sigma + 7.81$ **(6.1)**

For a 16 x 16 x 16 object space cube: $\chi = 6.25\sigma^2 + 11.25\sigma + 8.81$ **(6.2)**

For a 32 x 32 x 32 object space cube: $\chi = 18.75\sigma^2 + 13.75\sigma + 14.43$ **(6.3)**

With these equations, and using the σ_e estimation procedure[1], the BTF/FTB algorithm cutover point can be estimated independently by each SSG for a given set of user threshold window settings. Once σ is in hand, the appropriate equation from 6.1 - 6.3 is used to calculate χ. Then the cutting plane location is compared to χ, if the cutting plane is ahead of χ, then the adaptive FTB algorithm is used, otherwise the adaptive BTF algorithm is selected by the SSG.

Table 6.1
The Dynamic Hidden-Surface Removal Algorithm

1. Produce a histogram of $\{\mathcal{N}_o\}$ as the data volume is loaded.
2. After the data set is fully loaded, the percentage of the voxels in $\{\mathcal{N}_o\}$ having each density value is calculated using the histogram data.
3. For each set of user inputs, each SSG independently determines if its $\{\mathcal{N}_I\}$ lies in front of the cutting plane using the cutting plane intersection and back-to-front reference plane distance tests in the adaptive BTF HSR algorithm.
4. Each SSG lying in front of the cutting plane performs no computations and sends its MP a no-data message.
5. If an SSG is fully behind the cutting plane, the adaptive FTB algorithm is used by the SSG to generate its image space output.
6. If the cutting plane intersects an SSG's $\{\mathcal{N}_I\}$, the SSG computes σ_e by summing the histogram percentages and uses this figure to calculate χ using the proper equation from 6.1 - 6.3.
7. If the cutting plane lies behind χ, then the adaptive BTF algorithm is used, otherwise the adaptive FTB algorithm is used.

[1]To recap the procedure, σ_e is based upon the histogram of $\{\mathcal{N}_o\}$ found as each SSG loads its data set. The histogram is used to calculate each density value's percentage population within $\{\mathcal{N}_o\}$. The value for σ_e for a given threshold window is obtained by summing the density value percentage population values of the density values which lie within the threshold window.

Using the Dynamic Adaptive SCC HSR algorithm, a 64^3 voxel volume depicting a human vertebrate in gel was rendered using the hypercube interconnection topology with 64k byte message packets at 0° z-axis rotation. The resulting image is presented in Figure 6.27.

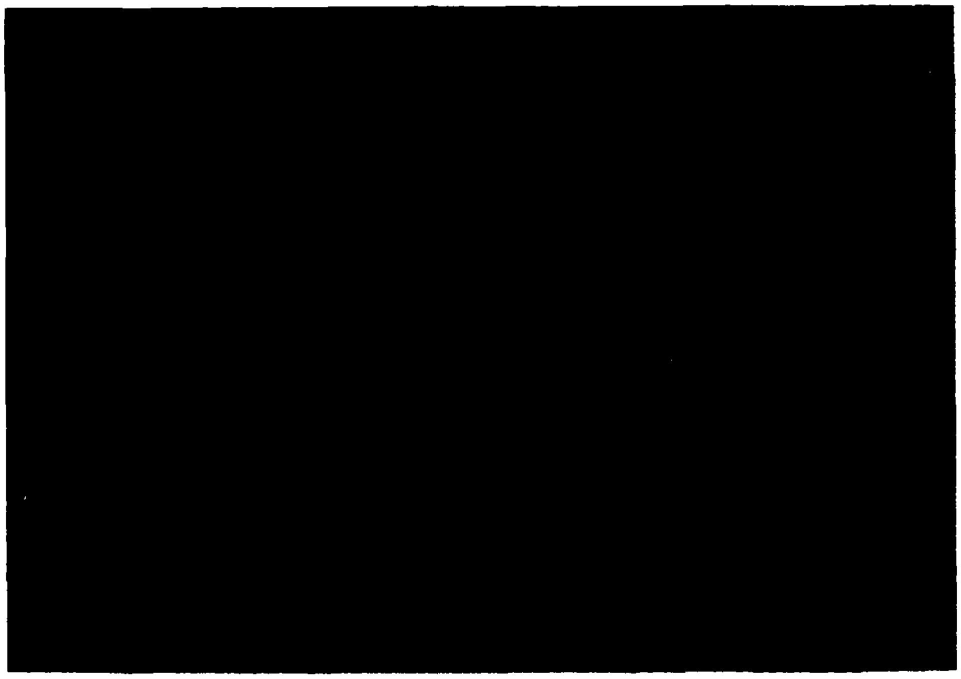


Figure 6.27: Human Vertebrate in Gel Rendered in the Hypercube Interconnection Topology Using the Dynamic Adaptive Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 0° z-axis Rotation at $\sigma = 100\%$.

Using the Dynamic Adaptive SCC HSR algorithm, a 64^3 voxel volume depicting a mitre box was rendered using the hypercube interconnection topology with 64k byte message packets at 0° rotation as well as 45° x-axis and 45° y-axis rotation. The resulting images are in Figures 6.28 and 6.29.

The use of the Dynamic Adaptive SCC HSR algorithm results in approximately a 50% performance improvement over the basic BTF SCC HSR algorithm. The range of performance varies from a 20% improvement at various rotations when $\sigma = 50\%$ and the cutting plane lies at the front of $\{\mathcal{N}_J\}$ to a 99% improvement for all σ values when the

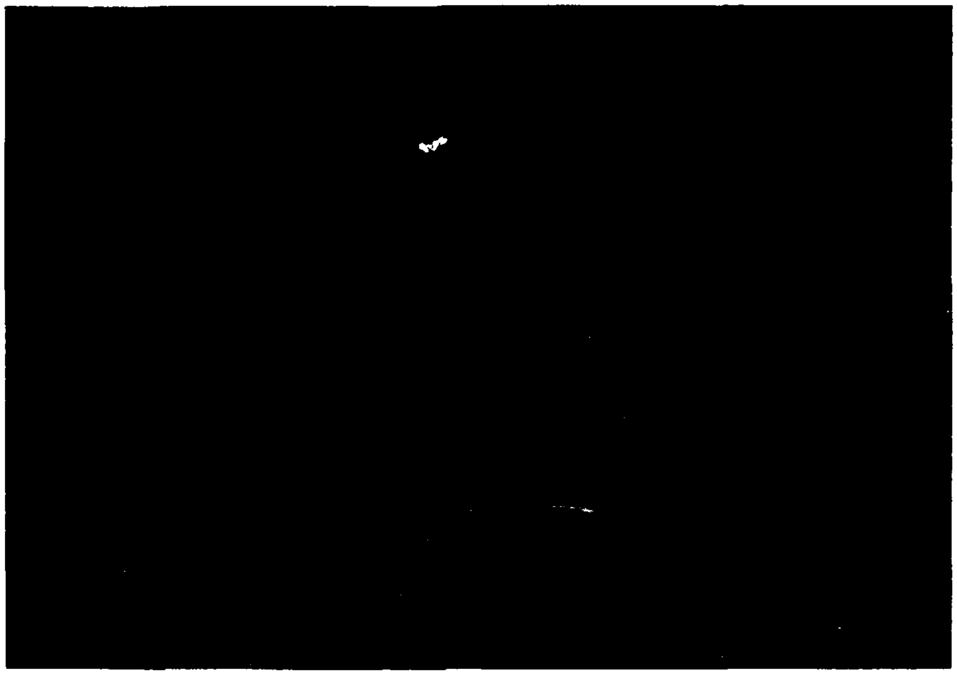


Figure 6.28: Mitre Box Rendered in the Hypercube Interconnection Topology Using the Dynamic Adaptive Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 0° z-axis Rotation and $\sigma = 100\%$.

cutting plane lies at the back of the $\{\mathcal{N}_J\}$ for an SSG. For illustrative purposes, the image rendering time performance obtained using the dynamic HSR algorithm on a 32^3 object space is compared to the basic, edit, and adaptive HSR algorithms Figures 6.30 - 6.31. Illustrative results for 8^3, 16^3, and 32^3 object space volumes across a range of σ values are provided in Appendix M.

To fully exploit this advance in SSG performance, a change to MP operation is advisable. In previous discussions of the medical imaging pipeline, MP operation required that the MP wait until all SSG inputs are available before merging the input data in either BTF or FTB order. With the adoption of the dynamic HSR algorithm by the SSGs, an MP may receive input data sets at irregular intervals instead of nearly simultaneously. Therefore, the MP is allowed to merge the first input data set it receives into the empty output buffer, and then continue merging later arriving data sets as they arrive so long as the incoming data set is either FTB or BTF adjacent to data already present in the output

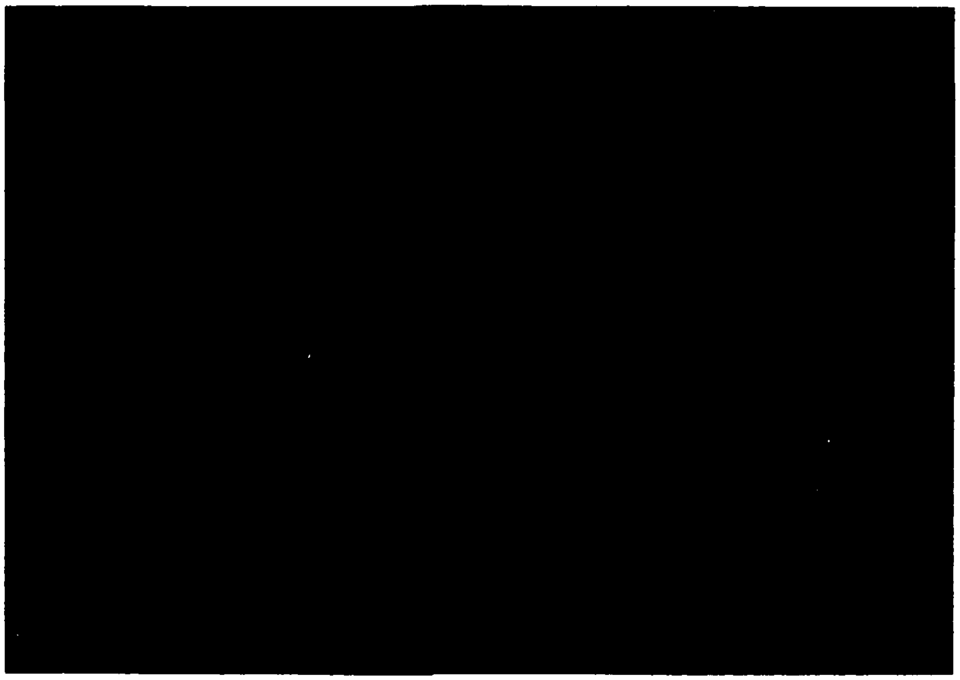


Figure 6.29: Mitre Box Rendered in the Hypercube Interconnection Topology Using the Dynamic Adaptive Subcube-Centered (SCC) Hidden-Surface Removal (HSR) Algorithm With 45° x-axis and 45° y-axis Rotation and σ = 100%.

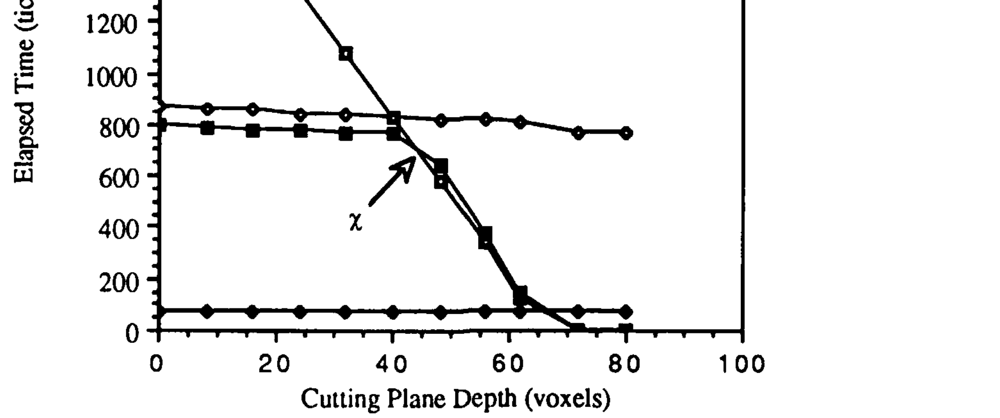


Figure 6.30: Image Rendering Elapsed Time vs Cutting Plane Location for the Dynamic, Basic, Adaptive and Edit FTB and BTF HSR Algorithms at σ = 100%.

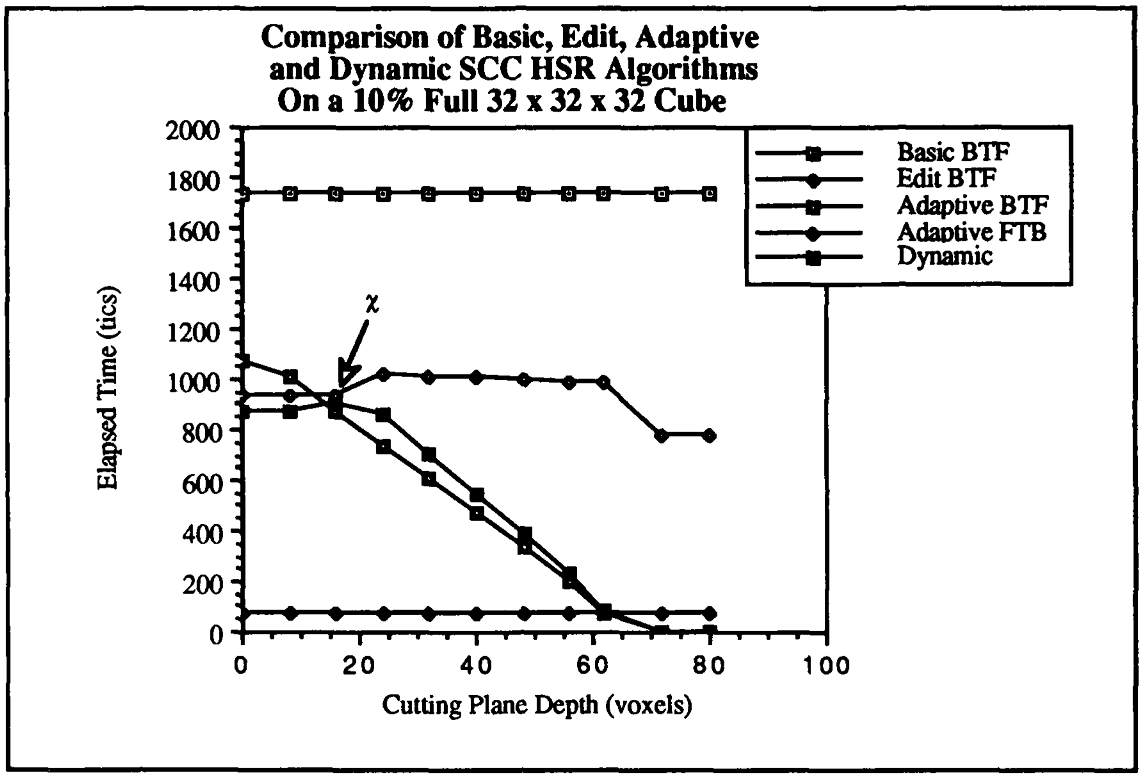


Figure 6.31: Image Rendering Elapsed Time vs Cutting Plane Location for the Dynamic, Basic, Adaptive and Edit Front-to-Back (FTB) and Back-to-Front (BTF) HSR Algorithms at $\sigma = 10\%$.

buffer of the MP. If a data set is not adjacent, it is held until it is and then it is merged. When an MP finishes its input data merge, the result is forwarded to the next stage, where that MP also employs the "merge 'em as you get 'em" strategy. The net effect is that T_e, the elapsed time to render the full image is decreased, as well as T_r, the rate as which images are produced.

The requirement in Chapter V for an MIMD multiprocessor is amplified with the use of the dynamic HSR algorithm because a SIMD implementation would be cumbersome and difficult, if not impossible to achieve because of the parallelism required. The medical imaging pipeline as described in Chapter V required MIMD parallelism because each stage of the pipeline performs different operations. With the use of the dynamic HSR algorithm, and the modification to MP operation described above, the processors in each stage now employ different algorithms, dynamically selected for each user input, providing loosely-coupled parallel operation along both axes of the pipeline. Since the processors in each stage can now optimize their performance based upon the unique processing situation local to each processor, a performance gain is achieved for the pipeline. Pipeline performance on a 32^3 image space for $\sigma = 100\%$ and $\sigma = 10\%$ with zero rotation and the cutting plane at the front of the volume employing the dynamic HSR algorithm in the SSG stage is compared to the performance obtained for the basic and edit HSR algorithms under identical conditions in Figures 6.32 and 6.33. These results demonstrate that the dynamic HSR algorithm provides a noticeable reduction in 3D image rendering time.

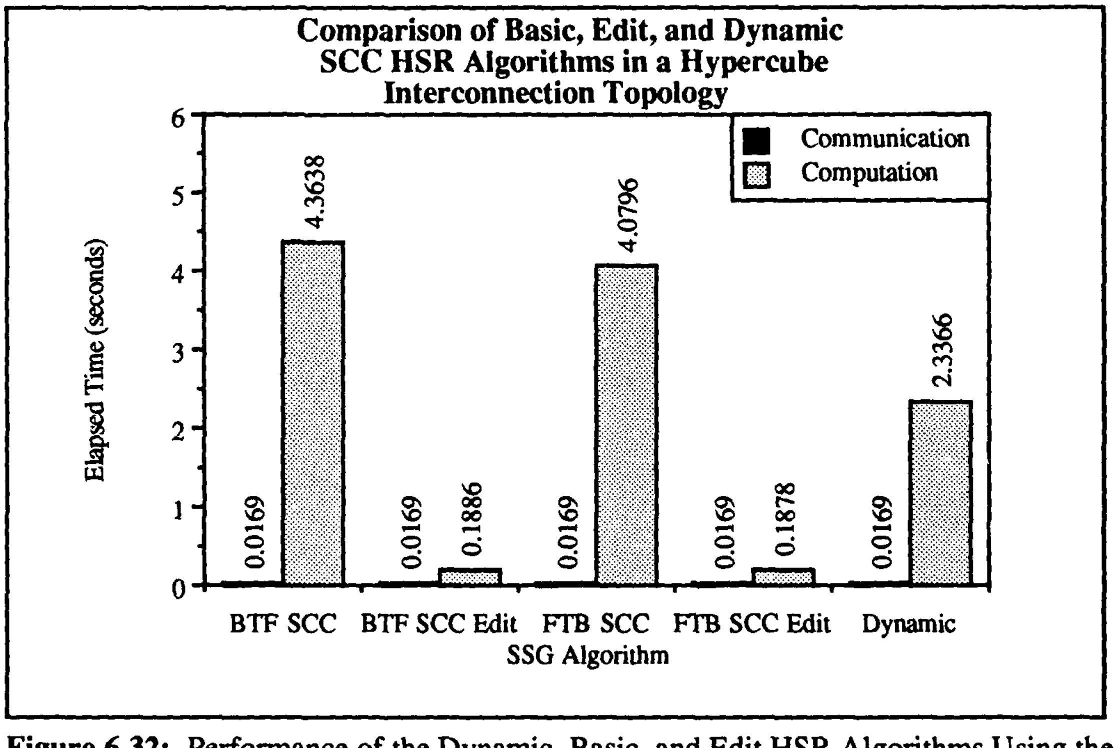


Figure 6.32: Performance of the Dynamic, Basic, and Edit HSR Algorithms Using the Hypercube Interconnection Topology at $\sigma = 100\%$ and with the Cutting Plane at the Front of the Volume.

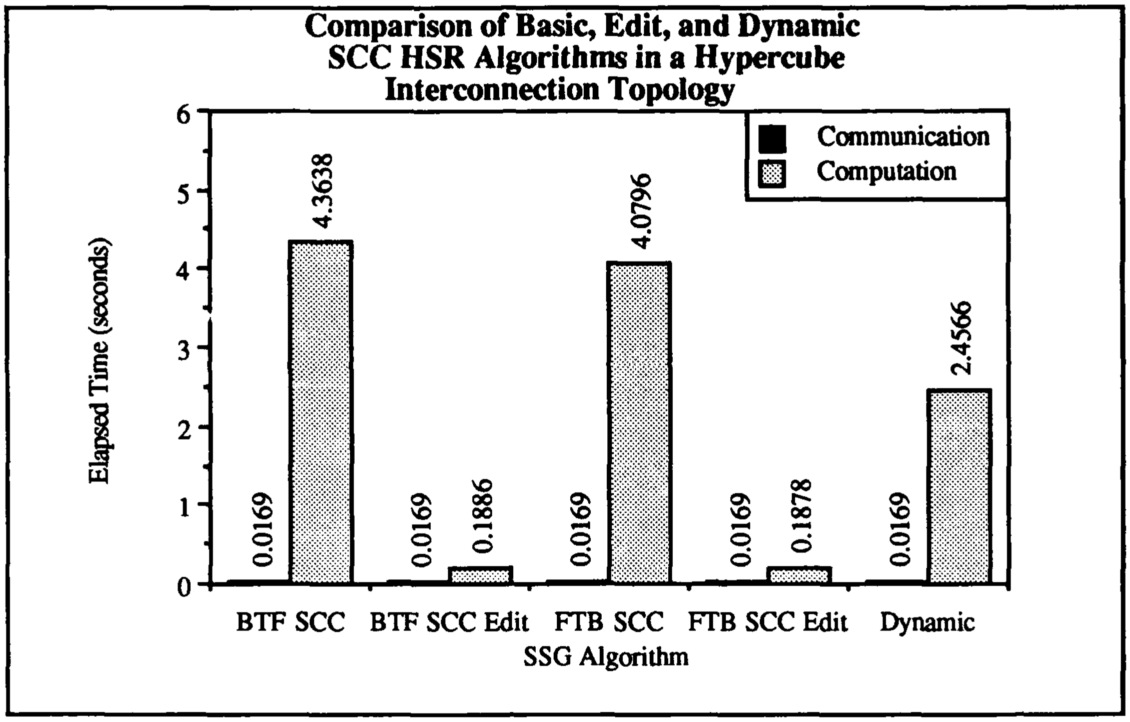


Figure 6.33: Performance of the Dynamic, Basic, and Edit HSR Algorithms using the Hypercube Interconnection Topology at $\sigma = 10\%$ and with the Cutting Plane at the Front of the Volume.

6.7 Summary.

In this chapter , two adaptive hidden-surface removal algorithms have been described and demonstrated to run faster than the basic BTF and FTB HSR algorithms for rendering 3D images. The dynamic HSR algorithm was then described. This algorithm capitalizes upon the strengths of the adaptive algorithms by allowing each SSG to select the algorithm which offers the best performance given the local processing situation at each SSG. The dynamic HSR algorithm provides an average 50% performance improvement over either of the basic HSR algorithms. As a means of further improving the performance of the medical imaging pipeline, the dynamic HSR algorithm can be combined with a change to MP operation to reduce T_e and T_r, and this combination was shown to provide superior performance to the medical imaging pipeline when running the basic HSR algorithm in the SSG stage and a single algorithm in the MP stage.

The next chapter presents broad conclusions and a summary of this work as well as outlining future avenues for investigation.

CHAPTER VII
CONCLUSIONS AND FUTURE WORK

7.1 Introduction.

The preceding pages outlined the development of a medical imaging machine which renders 3D images from raw voxel data within a Multiple-Instruction, Multiple-Data (MIMD) multiprocessor architecture at interactive rates. The interactive performance of the machine is achieved using a dynamic adaptive hidden-surface removal algorithm, which provides a 50% reduction in image rendering elapsed time over other recursive hidden-surface removal algorithms. This chapter recaps this research and suggests future work directed toward improving three-dimensional (3D) medical image quality and rendering speed.

7.2 Summary and Conclusions.

Chapters II - IV presented a comprehensive survey of the medical imaging, graphics, and medical-imaging modality literature. This survey was conducted with three purposes in mind. First, to determine the unique aspects of 3D medical imaging which impact 3D image rendering operations. Second, to ascertain the image processing techniques common to all 3D medical imaging machines. Third, to capitalize upon the experience of prior researchers to determine (1) which portions of the 3D image generation operation are parallelizable, and (2) the techniques applied in the past to achieve this parallelization.

Chapter II presented a description of five medical imaging modalities: Computerized Tomography, Magnetic Resonance Imaging, Positron Emission Tomography, Single Photon Emission Computed Tomography, and Ultrasound. The discussion in that chapter concentrated primarily on the imaging aspects of each modality such as resolution, limitations, and physical interpretation of the image data. In addition, the physics and operation of each modality was briefly sketched and the tomographic reconstruction process was described.

Chapter III was dedicated to a review of the graphics techniques and data models developed for 3D volume rendering, with particular emphasis placed on the techniques employed for rendering medical image volumes. The object and image space coordinate systems used in this dissertation were explained, and the octree, cuberille, and voxel data models were defined. An introduction to the ray tracing literature was presented. The chapter summarizes the techniques devised for performing the fundamental image processing tasks of geometric transformation, hidden-surface removal, shading, image segmentation, and anti-aliasing within the context of the voxel data model.

To provide a setting for the results presented in Chapters V and VI in terms of prior efforts in the 3D medical imaging arena, Chapter IV described eleven medical imaging machines. The major innovation(s) within each machine were characterized and the approach taken by each machine to the task of medical image rendering was examined in terms of the shading, anti-aliasing, and hidden-surface removal algorithms employed, the machine architecture, and the degree of parallelism in each machine. The performance of each machine was assessed in terms of image resolution and 3D medical image rendering speed. The material presented in Chapter IV allowed me to draw two conclusions of importance to my research. First, due to the amount of data to be processed, a pipeline approach to image processing is required if rapid image generation is to be achieved. This conclusion is supported by the fact that all ten machines designed to achieve an interactive image rendering capability employ some form of pipeline. Second, due to data independence in the voxel data model, the image data can be partitioned into smaller units for independent processing by individual processors (as in Pixel Planes, Reynolds' Voxel Processor, Kaufman's Cube machine, and Ardent's Titan). A further benefit which accrued from this search was the realization that no researcher or company has attempted to produce 3D medical images at interactive rates using a MIMD general-purpose medium-grain multiprocessor. Because the performance of this class of machines has improved dramatically over the last few years, I decided to investigate the suitability of this class of computing machine to the 3D medical image rendering task. Due to the decision to host the image rendering pipeline within a general-purpose machine, the image processing algorithms were implemented in software.

The host parallel processing machine is assumed to be a distributed memory MIMD (Multiple Instruction, Multiple Data [Fly72]) architecture machine using message packets for internode communication[1]. My MIMD multiprocessor-hosted pipeline approach segregates the image processing operations into functional units, facilitating the use of modular component design and testing via independent operation of the individual stages of the pipeline. In addition, the performance of the pipeline can be incrementally improved by attacking the performance of critical, bottleneck operations rather than addressing the image rendering task as an indivisible operation.

The motivation for adopting this MIMD based, pipeline approach comes from the twin requirements for image accuracy and interactive rendering speed as well as the desire for a flexible implementation of the image rendering algorithms. There are two drawbacks to the approach I chose, the slowness inherent in a software implementation and the large

[1] The use of message packets eliminates the memory contention/synchronization issues which affect the shared memory machines while permitting inter-process synchronization.

volume of data to be processed. To achieve interactive processing speeds within these constraints requires the distribution of the processing workload within a parallel processing environment.

The research presented in Chapters V and VI is directed at five distinct, yet interrelated research issues. The first concern was the proposal of an image processing pipeline and the selection of algorithms used within the pipeline for image generation. The second step addressed locating the processing bottlenecks in the image generation pipeline when it was embedded in a general-purpose MIMD multi-processor. The third issue was determining the cause of these bottlenecks and devising techniques for reducing/eliminating their impact. The fourth concern was formulation of a definition of recursive hidden-surface removal algorithms to guide the attack on the bottleneck in the Subscene Generator (SSG) portion of the pipeline. The fifth phase, which is the major portion of my work, addressed employing this definition to reduce/minimize the amount of time spent in the Subscene Generator stage bottleneck. The significant portions of my work in each of these topics is summarized in the following paragraphs.

Chapter V concentrated on the exposition of my work in the first four areas described above, with Chapter VI dedicated to the presentation of new algorithms which significantly reduce the computation required to form a 3D medical image.

Chapter V describes the five stage image processing pipeline which was designed based on the insight gained into 3D medical imaging via the survey. Processing within the pipeline is organized into a front- and back-end. The front-end of the pipeline performs the user interface and volume rendering operations in software upon the unprocessed voxel data. The back-end of the pipeline accomplishes pixel shading and 3D image enhancement operations in hardware. The use of software rather than hardware in the front-end of the pipeline provides a flexible, modular implementation of the image rendering algorithms. The decision to use raw voxel data, rather than preprocessing the data to extract surfaces, avoids the information loss inherent in the surface detection and surface tracking approaches to volume reduction. The use of hardware to perform the pixel shading operations provides the capability for high throughput in the portion of the pipeline where the computational workload demands processing speed.

The correct operation of the pipeline was validated in a trial implementation of a 3D image rendering machine, with recursive BTF/FTB hidden surface removal algorithms employed in the second, Subscene Generator, stage of the pipeline. The decision to use recursive back-to-front (BTF) and front-to-back (FTB) hidden-surface removal algorithms was based upon three advantages possessed by these algorithms within a voxel-based general-purpose MIMD multiprocessor-hosted image rendering pipeline. First, this class of algorithms combines hidden-surface removal, image rotation, and cutting-plane insertion into one operation, thereby minimizing the amount of processing performed. This reduction is especially important in a general-purpose machine because the algorithms are placed in software rather than hardware, thereby automatically increasing the amount of time required to form the image. Second, the algorithms are scalable across a wide range of image volumes, albeit with the restriction that the scene dimension be a power of two. Third, the algorithms support independent operation of the processors performing the hidden surface removal operation, which is an important consideration for this research as the image rendering operations are carried out within a general-purpose MIMD multiprocessor. A fourth advantage, which surfaced over the course of my research, is that these algorithms support both object and image space partitioning of the data volume.

Both object and image space partitioning are employed within the image rendering pipeline in order to better exploit the processing power of the MIMD multiprocessor. Image space partitioning is performed on the rendered volume, thereby reducing image rendering processing time by distributing image shading processing among several processors. A static, equal distribution of object space data among the pipeline front-end processors is used to achieve a balanced workload among the SSG processors in the pipeline. This partitioning of object space achieves equal image complexity, and hence processing load, among the processors in the SSG stage. In order to achieve acceptable elapsed rendering time performance in a MIMD multiprocessor employing message passing, the algorithms employed in the SSG processors must be able to operate autonomously, otherwise communication costs become prohibitive. This restriction is met by the use of recursive BTF/FTB hidden-surface removal algorithms in the SSG stage.

To determine which portions of this pipeline would be system bottlenecks in an actual MIMD multiprocessor, the implementation of the 3D medical imaging machine pipeline was hosted within simulated MIMD multiprocessors having hypercube and mesh processor interconnection topologies. The initial implementation employed the basic FTB/BTF hidden-surface removal algorithms, which are variations on the recursive hidden-surface algorithm described in [Rey85]. The accuracy of the images rendered with these, and subsequently developed hidden-surface removal algorithms, was verified using the techniques described in Chapter V. The hypercube and mesh interconnection topologies proved to be equally suitable processor interconnection schemes across a wide range of message packet sizes because the time required to move message packets through the pipeline within either topology is negligible compared to the accumulated computation time in the pipeline. Nevertheless, since the hypercube interconnection topology communication performance scales effectively to larger machines, I chose this topology as the host interconnection scheme for the work presented in Chapter VI.

The initial pipeline performance results supported my conclusion that the use of data partitioning in a parallel processing environment would be the key to interactive image rendering performance. Furthermore, as shown in Chapter V, this initial series of experiments revealed three bottlenecks in the image processing pipeline: image shading, scene formation at individual processors ($\{\mathcal{N}_I\}$ generation), and $\{\mathcal{N}_I\}$ merging into a final image. As the image shading bottleneck can be removed using special purpose hardware attached to a general purpose multiprocessor, I decided to attack the $\{\mathcal{N}_I\}$ generation bottleneck within the SSG stage of the pipeline, as this stage constitutes a critical pipeline bottleneck. The other 3D medical imaging systems described in Chapter IV also spent considerable effort in reducing the time spent on the processing performed in this stage, so the performance results and conclusion regarding the focus of my research effort is born out by other researchers in the field.

To guide the effort to reduce the amount of processing performed in the Subscene Generator stage, a definition of recursive BTF/FTB algorithms was developed and described in Chapter V. Based on this definition, the first step toward reducing the time spent in the SSG stage was taken in the development of BTF and FTB algorithms suitable for scene editing, ie., the image they form is only of that portion of the scene which lies on the cutting plane and the remainder of the image which lies behind the plane is ignored. The resulting picture set is analogous to a series of tomographic scans taken at the desired orientation. These algorithms are acceptable for rapid examination of the scene, but as these algorithms do not provide true 3D images they are not acceptable for 3D diagnostic viewing. A description of these algorithms and their performance within the simulated hypercube and mesh multiprocessor interconnection topologies is contained within Chapter V.

Two approaches to achieving a reduction of 3D image rendering elapsed time were considered at this point, development of completely new algorithms for $\{\mathcal{N}_I\}$ generation or modification of existing algorithms to exploit the MIMD architecture. Because of the advantages of recursive BTF and FTB algorithms, reduction of rendering time through modification of existing FTB and BTF HSR algorithms was attempted and is described in full in Chapter VI and summarized in the following paragraphs.

To reduce image processing time for BTF algorithms, a processing termination capability was developed. The operation of the adaptive recursive BTF hidden-surface removal algorithm is based upon information concerning the distance between the current P_I and the cutting plane as well as the distance from the current P_I to the furthest point in the current $\{\mathcal{N}_I\}$ and the Back-to-Front Reference Plane. This information allows each

SSG to individually and selectively terminate BTF processing within each of its octants, and recursively each suboctant, as soon as it begins processing data which lies in front of the cutting plane. This algorithm significantly reduces image rendering time when the cutting plane is positioned toward the rear of an SSG's $\{\mathcal{N}_J\}$, but when the cutting plane is at the front of the $\{\mathcal{N}_J\}$, the rendering time performance is worse than that of the basic recursive BTF algorithm.

A different approach to image rendering time performance improvement was taken for recursive FTB algorithms. For FTB algorithms, image processing time can be reduced by determining the number and location of the coordinates each SSG can write in screen space and then allowing each SSG to terminate processing as soon as these coordinates are written. The number and location of the screen space coordinates is determined by projecting each SSG's $\{\mathcal{N}_J\}$ into screen space and recording the coordinates and total number of pixels predicted to be written by each recursively smaller suboctant within the corresponding node of an oct-tree. The adaptive FTB algorithm operates directly on the voxel data without a boundary detection preprocessing step, thereby reducing the total amount of processing required by each SSG, and therefore the pipeline, to form an image. The adaptive FTB recursive hidden-surface removal algorithm exhibits elapsed image rendering time performance which is consistently superior to that of the basic recursive FTB hidden-surface removal algorithm. However, the performance of this algorithm does vary somewhat with the location of the cutting plane within each SSG's $\{\mathcal{N}_J\}$ and varies significantly with the threshold window saturation at each SSG.

At this point, it became clear that the MIMD multiprocessor environment permitted a unique approach to reduction of elapsed image rendering time: the dynamic selection by each SSG of either the adaptive BTF or FTB algorithm based on the processing situation faced by each SSG. The dynamic hidden-surface removal algorithm is described in detail in Chapter VI and the performance of the algorithm is demonstrated there. Recall that the pipeline was embedded in a MIMD machine with different stages employing different algorithms, but the processing flexibility offered by a MIMD machine was not fully exploited since the processors in each stage employ the same hidden-surface removal algorithm. As shown in Chapter IV, other parallel-architecture-based machines use a SIMD architecture at each stage of the image processing pipeline as this is one way to reduce the amount of time spent producing the image. These machines relied upon rapid execution of the algorithms encoded in the hardware of the machine for their speed. My choice of a general-purpose MIMD multiprocessor as the assumed host machine provided the impetus for examining a new approach to volume rendering at an interactive rate.

Experimentation revealed that for a given set of rotations, local threshold window saturation, and cutting plane penetration, the processing situation can vary greatly from SSG to SSG. Some SSGs may be completely in front of the cutting plane, some may have the cutting plane pass through the forward portion of their space, some may have the cutting plane pass through the rear portion of their space, and some may have the cutting plane completely in front of *their* $\{\mathcal{N}_J\}$. *Threshold saturation can vary from 0% to 100%.* Dynamic algorithm selection reduces the amount of time each SSG spends processing its scene, and so reduces the amount of time spent in this stage, by considering these factors before selection of a hidden-surface removal algorithm. In order to completely exploit the possibilities offered by dynamic algorithm selection, a fully MIMD machine is required to support operation of the image rendering pipeline. The algorithm selection criteria I developed rely on the orientation of the scene, the local threshold window saturation, and the cutting plane location within the $\{\mathcal{N}_J\}$ of each SSG to determine the algorithm for each SSG which promises the lowest processing time. Dynamic algorithm selection therefore reduces the amount of time spent in the SSG stage of the image rendering pipeline by providing the performance of the adaptive FTB hidden-surface removal algorithm when that algorithm is optimal. However, any SSG can transparently switch to the adaptive BTF algorithm when that algorithm promises to reduce local image rendering time.

Given the dynamic adaptive HSR algorithm, the final questions to be addressed concern the capabilities required of the multiprocessor host machine and the performance provided by my medical imaging pipeline when running the dynamic adaptive HSR algorithm. Assuming a 1 MIP processor at each node of the multiprocessor, a hypercube interconnection topology, and a medical image size of 256 x 256 x 128 voxels, the host requirements and nominal performance of the pipeline for two SSG object space volumes, 16 x 16 x 16 voxels and 32 x 32 x 32 voxels, are presented in Table 7.1.

Table 7.1
Multiprocessor Host Requirements for and Representative Performance of the Medical Imaging Pipeline

SSG Object Space Volume (voxels)	Total SSGs Required	Total Processors Required	Memory Requirement per Processor	T_e (seconds)	T_r (seconds)
16 x 16 x 16	2048	2341	1 Mbyte	3.1	1.9
32 x 32 x 32	256	293	5 Mbytes	17.6	15.1

These memory and processor requirements, which are by no means modest, can be met by currently available commercial multiprocessor machines.

7.3 Future Work

This section presents four proposals for research involving ray tracing within MIMD multiprocessors, improving the photo-realism of the dynamic hidden-surface removal algorithm, image segmentation, and interactive volume registration.

The image processing pipeline described in this dissertation can be adapted to perform 3D image rendering via ray tracing. My comments here are limited to a few initial observations on the implementation of this conversion. The key requirement for incorporating ray tracing into the image rendering pipeline is developing a technique for ray propagation which equally distributes the computational burden among the processors in the SSG stage and minimizes communication costs. One technique, proposed by Dippe' in [Dip84] and designed for multiprocessors, is to begin with an approximately equal partitioning of object space among the processors and then adaptively re-allocate object space as the workload at each processor is characterized. This approach propagates rays through the volume by passing messages between processors, a potentially costly approach, and imposes additional computational overhead by requiring run-time characterization of the computational workload at each processor.

As an alternative approach, amenable to incorporation into the image rendering pipeline I proposed, I suggest that the object space partitioning should remain static and that the algorithms employed at each SSG should vary. This would eliminate the overhead required to characterize the processor workload at run-time and allow each SSG to reduce its image rendering time. The ray tracing algorithms can be selected from among those proposed by Levoy, Glassner, Fujimoto, or others, as the performance of these algorithms varies with the scene complexity. The only selection requirement is that the suite of algorithms all generate the same type of image. Selection of the algorithm to be used requires characterization of the image space volume at each SSG, and could possibly be accomplished using the σ_e operator I devised. In order to perform ray-propagation at minimal communication cost, ray propagation between SSGs can be simulated either by compositing or FTB overlaying in the Merge Processors (MPs). This simulated propagation technique is feasible because the 3D medical image rendering operation does not require spawning rays for reflection, refraction, or shadow determination. Volume rendering within each SSG would proceed as follows. Each SSG would first determine the appropriate ray tracing algorithm. Then, each SSG would establish the portion of screen space it is responsible for filling by employing the image space pre-projection portion of the adaptive FTB algorithm to compute the screen coordinates it will write to.

This second step gives each SSG a virtual screen and pixels from which to cast rays. Then, instead of applying a recursive hidden-surface removal algorithm, each SSG casts one or more rays from each screen space coordinate into its $\{\mathcal{N}_o\}$ volume.

As each ray progresses, the SSG computes the final value of the ray based upon the class of ray tracing algorithms implemented in the SSGs: volumetric compositing, halting upon encountering a density of a given value, halting when rgbα reaches a certain value, etc. The value for each ray is then stored within a 2D array, and resampled if necessary, to form the final output image from the SSG. The remainder of the volume rendering operation, which is carried out in the MPs, combines the SSG output volumes into a final image. If the final image is to display the foremost point at which each ray achieves a given value, FTB overlay with conditional summing of pixel values is performed. If volumetric compositing is being performed, the rgbα values are summed within the MPs. This pipeline-based ray tracing approach has the potential to eliminate the requirement for propagating rays from processor to processor, which would impose prohibitive communication costs, as well as providing the photo-realistic image quality provided by ray tracing.

A conservative approach to improving image quality while retaining the rendering speed of the volume rendering approach would build upon the dynamic recursive hidden-surface removal algorithm. The dynamic hidden-surface removal algorithm would render the volume normally except that the full-scale image space coordinates are used as weights to shade the integer screen space coordinates. Recall that the coordinate values of the $\{\mathcal{V}_I\}$ and the P_I are left shifted before they are used to generate $\{\mathcal{N}_I\}$. The computed coordinate values are then right-shifted to scale them back to values which lie within the image space volume. However, the magnitude of the number shifted out indicates the overlap of the object space voxel upon the surrounding four screen space pixels. Instead of discarding this information, it could be used to adjust the gray-scale value of surrounding pixels to reflect the overlap of each voxel. An additional benefit of this approach is elimination of the requirement for performing anti-aliasing by supersampling because anti-aliasing is accomplished implicitly via the gray-scale weighting function.

Although image segmentation was not employed within the volume rendering pipeline I use, segmentation is an important capability for a medical image rendering machine. The chief drawbacks to image segmentation as currently performed are its computational cost and the image information loss which may occur. To date, quantification of the information lost during segmentation has not been attempted, so I believe that significant benefit to the medical image rendering community would occur from efforts in this area. The issue of computational cost can be addressed via the use of MIMD multiprocessors to perform segmentation. Here, again, the concept of dynamic algorithm selection may prove useful. The medical imaging literature abounds with approaches to segmentation, ranging from thresholding to surface tracking. Characterization of the input volume at each processor, coupled with the development/adaptation of two or more segmentation algorithms from which to chose, may yield considerable computational savings. An additional concern is the autonomy of the segmentation algorithm. Currently, no approach is completely automatic; some operator intervention is required in order to correctly extract the volume. Clearly, the benefits of a multiprocessor approach are lost if the operation of the segmentation algorithm within tens, hundreds, or thousands of processors must be manually directed. Additional work in this area in the form of development of heuristic organ models and surface tracking/extraction techniques are needed.

A final proposal for future research lies is the use of large multiprocessors to perform interactive image registration. Presently, image registration is accomplished using post-hoc boundary fitting algorithms to align the volumes acquired at different times and/or using different modalities. While some progress has been made in this area, the image registration process is still batch-job oriented because of the extremely large data volumes which are manipulated and the requirement for registering each volume against all other volumes using a contour fitting approach. The use of a multiprocessor may allow the initial contour fitting computations to be performed rapidly. An additional step toward achieving an interactive capability with improved registration accuracy can be made by employing a large multiprocessor (8k nodes) for interactive image registration through simultaneous image generation from different subcubes of the multiprocessor. This approach allows each image to be independently rotated and aligned to the other images displayed, with the accuracy of the alignment dependent upon the display quality and the fit required by the operator.

sic finitum est

APPENDIX A

APPENDIX A

DEFINITIONS

The following graphics definitions are based on the CORE graphics standard nomenclature. Definitions which are not related to image processing are based on standard usage.

1D - one-dimensional.

2D - two-dimensional.

3D - three-dimensional.

Aliasing - an imaging artifact caused by the caused by undersampling of the image. Commonly seen as the jagged appearance of slanted lines. Because the frequency components of the image which occur above the Nyquist rate cannot be reconstructed at their true frequency, they are manifest as lower-frequency aliases of the original signal.

Alpha channel - a data structure used to provide the information needed to compose separately rendered images.

Antialiasing - any method which reduces the effects of aliasing.

Area/distributed light source - any light source which is not a point light source.

Axial view - a slice taken parallel to the patient's long axis and perpendicular to the patient's side-to-side axis and front-to-back axis, a frontal or head-on view.

β^+ particle - also called a positron, emitted during β^+ decay of a nucleus and is identical in all respects to an electron except that it has a electric charge of +1 instead of the -1 charge of the electron, making the positron the antiparticle of the electron. In β^+ decay, a proton in the nucleus is converted into a neutron and the excess energy is emitted from the nucleus as a positron and a neutrino. Used in PET to generate γ-rays through positron/electron annihilation.

Beam hardening - the preferential absorption (attenuation) of low-energy x-ray beams as they traverse a body which results in the increase of the average strength of the x-ray beam. This effect occurs as a result of using polychromatic (wide range of energies) x-rays instead of monochromatic x-rays for imaging.

BGO - Bismuth Germanate.

Bicubic surface patch - a parametric surface patch in which the equations defining $X = X(u,v)$, $Y = Y(u,v)$, and $Z = Z(u,v)$ are cubic in u and v.

Bq - becquerel, a unit of measure for radioactive decay, defined to be 1 disintegration per second.

BTF - back-to-front .

χ - back-to-front/front-to-back HSR algorithms cutover depth.

Ci - curie, a unit of measure for radioactive decay, defined to be 3.7×10^{10} disintegrations/second.

Clipping - the process whereby the portion of world coordinates not within a display window is identified so that it will not be mapped to the screen.

Coherence - a characteristic of an image/volume in which the properties of pixels/voxels adjacent in the x and/or y and/or z directions are similar. Because of this similarity, the properties of adjacent pixels are capable of being computed from the given pixel value. Employed for visible-surface determination in area-, line-, frame-to-frame, and object-coherence based algorithms.

Compositing - combining several rendered images into a single image.

Coronal view - a slice taken perpendicular to the patient's long axis and front-to-back axis and parallel to the side-to-side axis, a top-down view of a cross-section. Also called a <u>transverse</u> view.

CT - Computerized X-ray Tomography.

Cuberilles - small, cubical, closely spaced volumes in object space. Each cuberille is assigned a value based upon the physical property measured by the imaging modality. Commonly derived from interpolated voxel data.

Detected surface - the output of a medical imaging boundary detection program.

Diffuse reflection - the reflection component which represents the quantity of light scattered equally in all directions, therefore the surface will appear to have the same brightness from all viewing angles. For these surfaces, Lambert's cosine law relates the amount of reflected light to the intensity of the light source and the cosine of the angle between the surface normal and the point light source.

Displayed detected surface - the 2D picture of the detected surface in which each visible surface element is shaded according to an illumination model.

Dissection - the appearance of peeling away of overlying layers of tissue.

Dissolution - the appearance of tissue dissolving, or melting, away.

Extent - the volume of space occupied by an object.

Filtering - any process which reduces the high frequency components of an image. Employed to reduce the effects of aliasing.

FLL - front-lower left, location of the coordinate set within the octant.

FTB - front-to-back.

γ (MRI) - gyromagnetic ratio, used to describe the strength of the RF signal produced by an element in response to an external magnetic field.

γ (SPECT and PET) - the excess energy in the nucleus released during radioactive decay. Gamma decay can take one of two forms, emission of a high-energy photon or through electronic conversion, but only the photon emission process is relevant to PET and SPECT imaging. A γ ray and an x-ray of the same energy interact with matter in the same manner.

Geometrical transformations - mathematical operations by which the location, orientation, and size of the objects to be displayed are altered.

Hidden-surface removal - the process of determining the portions of a volume which are not visible (obscured) and deleting them from the rendered image.

HU - Hounsfield Unit, a unit of measure which represents the amount of X-ray attenuation within a scanned voxel volume relative to the X-ray attenuation which occurs in the identical voxel volume filled with water. The HU range is -1000 to +3000.

Hypercube - an n-cube. An n-cube is defined recursively as follows: Let the 1-cube $Q_1=K_2$. Then $Q_n = K_2 \times Q_{n-1}$. Thus, Q_n has 2^n points which may be labeled as $a_1a_2...a_n$, where each a_i is either a zero or a one. Two points of Q_n are adjacent if their binary labels differ in exactly one place. Figure A.1 contains diagrams of a 1-cube, a 2-cube, and a 3-cube.

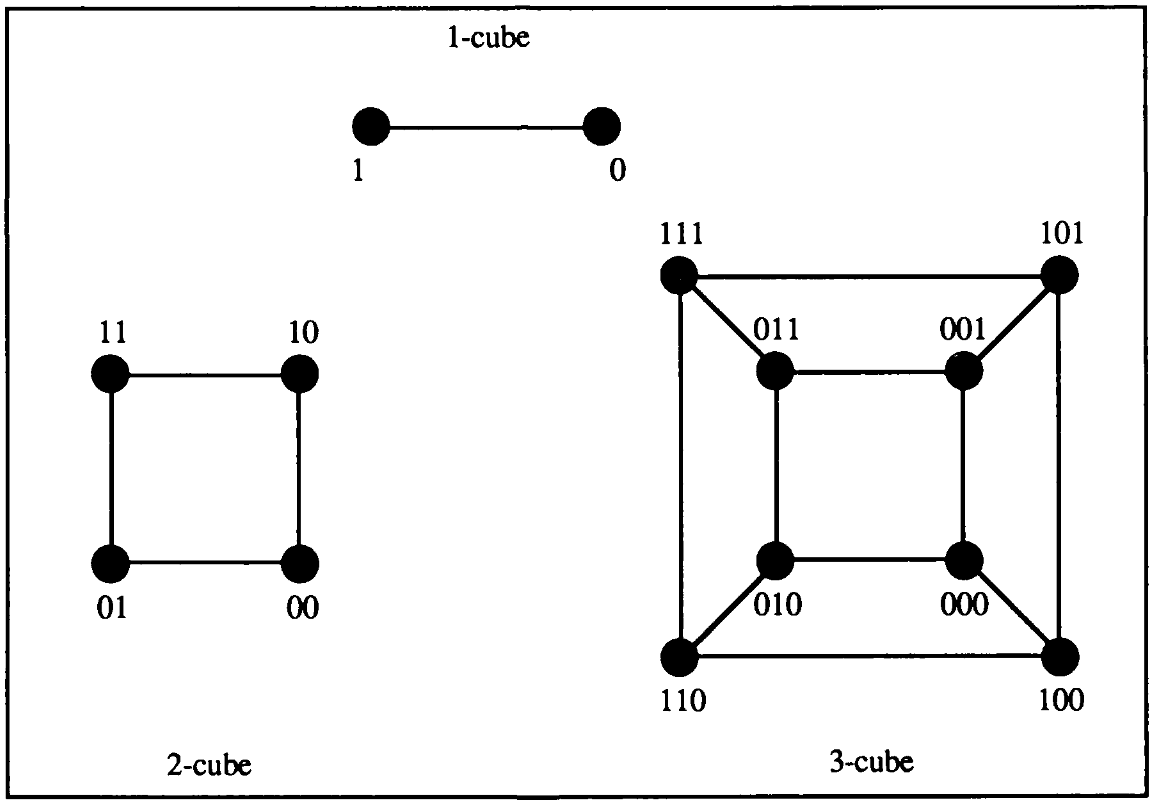


Figure A.1: 1-cube, 2-cube, and 3-cube

Illumination model - also called a shading model. The model attempts to depict the amount of light which reaches the eye of an observer from each point in the scene based on the observer's position, number, location, and color of light sources, the location of objects in the volume, and the surface properties of the objects. There are two broad classes of models, local models which model surface effects based on light and surface orientation and the global models which include reflection, shadows, and refraction effects.

Image space - the three-dimensional continuous space defined by the real-valued x', y', and z' axis coordinate system which contains the transformed object space volume .

Interpolation - produces a scene in which the voxel size is different from that in the original scene. The densities assigned to the new voxels are estimated based on the densities of the original voxels.

keV - thousand electron volts.

λ - the probability of decay per unit time for a single atom of a radioactive material.

Lamour frequency - the in-phase precession frequency of the nuclei of identical atomic mass which have been placed in a strong magnetic field, symbolized by w_0.

Low-pass filtering - the suppression of the high frequency components of the image. Intuitively, this type of filter removes the components of the image which change in less than a diameter of a pixel.

Mach band effect - the name given to the lines in an image that are produced along the intensity discontinuities which occur at a rapid change in the slope of the intensity curve in a section of an image.

Mobile proton density - a measure of the amount of free, or mobile, hydrogen that is available in the body for generation of the MRI signal, also called spin density.

MRI - Magnetic Resonance Imaging.

NaI - Sodium Iodide.

$\{\mathcal{N}_O\}$ - a set of neighborhoods of points in object space such that each neighborhood is a regular shape and the set of neighborhoods completely tessellates object space. $\{\mathcal{N}_I\}$ performs the same function in image space.

NURBS - non-uniform rational B-spline surfaces.

Nyquist limit - the highest frequency at which an image can be sampled without aliasing occurring in the reconstructed image.

Object representation - the data structure employed to store a modality's output image data within a medical imaging machine.

Object space - the three-dimensional continuous space defined by the real-valued x, y, and z axis coordinate system which contains the original volume/image.

Octree - the 3D analog of a quadtree and is constructed in a similar manner.

Oct-tree - a data structure which provides a hierarchical encapsulation of the spatial relationships between voxels in a volume.

Organ surface - the surface of the original organ *in vivo*.

Painter's algorithm - an algorithm which employs the hidden-surface removal strategy of overwriting the distant portions of the scene by those portions which are closer to the viewer.

Parametric surface - surfaces defined by three bivariate functions X = X(u,v), Y = Y(u,v), Z = Z(u,v).

Partial volume effect - a spatial aliasing artifact caused by the fact that a voxel's value represents a weighted integral over a volume of tissues. This effect can be offset by higher resolution in the data collection modality or by voxel neighborhood examination.

PET - Positron Emission Tomography.

PHIGS+ (Programmers' Hierarchical Interactive Graphics System) - an extension to the ANSI standard graphics committee's current PHIGS 3D graphics standard which supports lighting, shading, complex primitives, and primitive attributes. A PHIGS tutorial can be found in [Cah86].

Photo-realistic image - a computer-generated image which has the display characteristics, such as specular reflectance, shadows, and diffuse reflection, which would appear in a photograph.

Pixel - abbreviation for the term *pi*x*el el*ement. A single pixel is the smallest element of a picture that can be displayed, it is essentially a single point on the display screen.

PM - photomultiplier tube, used to amplify the light emitted by the γ-ray detectors in a SPECT or PET imaging machine.

$\{\mathcal{P}_O\}$ - the set of object space points upon which the generation of all the points in each $\{\mathcal{N}_O\}$ in the space is based. Each of the P_0 in $\{\mathcal{P}_O\}$ is selected from a different $\{\mathcal{N}_O\}$. $\{\mathcal{P}_I\}$ performs the same function in image space.

Point sampling - an image rendering technique which which samples object space on a point-by-point, rather than voxel-by-voxel, basis. This method of sampling typically results in aliasing unless several samples are taken of each volume.

Procedure inlining - a compilation technique which eliminates procedure calls by moving each called procedure to the point in the code at which it is called, thereby placing it in-line for better analysis for vectorization opportunities.

Projections - used to map objects in a 3D window onto a 2D viewport. The operation requires determining which portions of the object are visible and then mapping these portions to the viewport.

PZT - zirconate titanate ceramic.

Quadtree - a tree-based representation of a 2D space wherein each leaf node depicts a homogeneous area of the space. A quadtree is constructed in the following manner. Start with an image and check to see if it has a simple description. If it does, label a single node with the description and stop. Otherwise, divide the image space into four congruent square regions (called quadrants) and examine each to determine if it has a simple description. If any quadrant has a simple description, create a leaf node and label it with the description. If the quadrant does not have a simple description, create a branch node and create four new quadrants from the old quadrant, then repeat the test. Stop when each area in the image is represented by a leaf node.

Quantitation - the process of making measurements on organs and organ systems.

$\mathcal{R}$- a relation which, when applied to $\{\mathcal{V}_0\}$ and based upon an element of $\{\mathcal{P}_0\}$, generates the $\{\mathcal{N}_0\}$ which surrounds the P_0. Also used to generate $\{\mathcal{N}_I\}$ based upon $\{\mathcal{P}_I\}$ and $\{\mathcal{V}_I\}$.

Radiosity lighting model - accounts for diffuse inter-reflection between items within an image.

Raster scan display - maintains the display primitives such as lines, characters, and solid areas in a refresh buffer in terms of their component points, called pixels. The image is formed from the raster, which is a set of horizontal raster lines made up of individual pixels. The raster is simply a matrix of pixels covering the entire screen area, and the matrix is kept in a bit map maintained in a refresh buffer. The entire image is scanned out from memory 30 times per second, one raster line at a time, top to bottom.

Ray tracing - the performance of visible-surface determination by tracing individual rays through a scene.

Real-time image display - defined to be a display rate at or above the flicker-fusion rate of the human eye, which is in the range of 25 - 30 frames per second.

Reflection - light bouncing off a surface of an object.

Refraction - light bending as it passes through a transparent object.

Rendering - the process of converting a volume description in object space to a volume description in image space and then into an individual pixel color description in screen space.

RF - radiofrequency.

Sagittal view - a slice taken parallel to the patient's front-to-back axis and perpendicular to the long axis and side-to-side axis, a side view.

Scan conversion - the process of transforming an object description in image space to a pixel color description in screen space.

SCC - sub-cube center, location of the coordinate set within the octant.

Scene element - a primitive data element which represents some discrete portion of the volume which was imaged.

Screen space - the space defined by the image display device defined by the integer-valued u and v axis coordinate system wherein the integer u and v coordinates correspond to pixel locations.

Segment - a logically related collection of output primitives which describe a portion of an object. The segments of an object are used to reconstruct an image of an object.

Segmentation of a scene V, where V=(S,F) - a mapping from V into a binary scene V'=(S,F'). S = {v | v within the window}, F is a mapping from S into a set of voxel density values, and F' is a mapping from S into a set of pixel gray-scale values. The original and resulting scenes are of the same size and are sub-divided into voxels in the same way. Thresholding is an example of segmentation.

Shading - a graphics process whereby the appearance of a visible surface is altered to account for the number and types of light sources illuminating the surface, the surface texture, color, and reflectance, and the position and orientation of surrounding surfaces. The shading algorithm employed for a scene is based on an illumination model.

σ - threshold window saturation, the percentage of voxels within $\{\mathcal{N}_0\}$ which fall within a given threshold window.

Slice of a scene - the scene formed by all voxels (i, j, k) for a fixed k. The 2D portion of the scene thus extracted is referred to as the k'th slice.

SPECT - Single Photon Emission Computed Tomography.

Specular reflection - reflection caused by light reflection from the outer surface of an object which typically appears as a highlight on the surface of the object. Specular reflection is observed only for that portion of the surface for which the angle α between the reflected light and the direction to the viewer is near zero. Specular reflection can be modeled by $\cos^n\alpha$, where n varies between 1 and 200 depending on the surface.

Spin density - a measure of the amount of free, or mobile, hydrogen that is available in the body for generation of the MRI signal, also called mobile proton density.

Stochastic sampling - sampling of an image/volume at nonuniformly spaced locations instead of on a regularly spaced grid.

Straight ray assumption - the assumption (made for computational tractability) that ultrasound waves travel in a straight line from source to detector without diffraction, refraction, or multiple reflections.

Supersampling - a technique in which more than one volume sample is taken per pixel at regular intervals across the volume. Used to offset the effects of aliasing in an image.

T_1 - the length of time required for the energy injected into the nuclei during the RF pulse phase of an MRI study to fade away. Equivalently, it is also the time required for the protons in the nuclei to return to thermal equilibrium with their surroundings by re-aligning with the strong external magnetic field.

$T_{1/2}$ - radioactive half-life.

T_2 - the theoretical amount of time that the signal from the most recent RF pulse of an MRI study remains detectable.

T_2* - the actual amount of time it takes a MRI signal to disappear for a given piece of equipment. This time is related to the quality of the magnets used to impose the magnetic field.

T_e - elapsed time, the amount of time required to convert a user input to a rendered an image.

Tesla - 10^4 Gauss, the strength of the Earth's magnetic field is 2×10^{-5} Tesla

Thread - a locus of control within a process which consists of a program counter and the register states of the thread. The threads of a process share address space and OS context but their registers and stack space are private.

Tomography - the representation of a slice. Implies the formation of 2D cross-sectional images of a portion of a body which are free from interference from structures not in the plane of interest. The images thus formed are called tomograms. From the Greek *tomo* for slice.

$\mathbf{T_r}$ - the rate at which images emerge from the medical imaging pipeline.

Transparency - a material property which allows light to be transmitted through the material.

Transverse view - a slice taken perpendicular to the patient's long axis and front-to-back axis and parallel to the side-to-side axis, a top-down view of a cross-section. Also called a coronal view.

Viewport - a rectangular portion of the screen onto which the window is mapped. A viewport is a logical output device whose normalized device coordinates are specified in the range 0-1 for both the x and y directions on the screen.

View volume - the 3D section of the world which is to be displayed.

$\{\mathcal{V}_O\}$ - a set of vectors such that each vector is oriented from the object space origin to a different element of $\{\mathcal{P}_O\}$. $\{\mathcal{V}_I\}$ performs the same function in image space.

Volumetric compositing - any image rendering technique based upon blending of semitransparent voxels such that all the voxels along a beam or ray originating from a given pixel contribute to the final color of the pixel.

Volume rendering - any image generation technique which does not explicitly extract contours or surfaces of an object to be displayed but processes all the voxels (cuberilles) within the scene at display time (first defined in [Rey85]). A key characteristic of this approach is that all of the original data are available at all times for generating the image.

Voxel - a parallelpiped shaped volume element. This three-dimensional analog of a pixel formed by dividing object space with sets of planes parallel to each object space axis. A three dimensional set of voxels will, therefore, represent the entire volume. Voxels have four characteristic properties: they are all of the same size, they are discrete, they lie along three mutually perpendicular axes, and they are very small relative to the object being imaged.

Voxel density - the integer value assigned to a voxel that corresponds to a parameter in the volume that the voxel represents.

Window - the rectangular region in world coordinates that is to be displayed.

X-rays - a stream of high energy photons with a wavelength between 0.05 angstroms and 100 angstroms.

X Window system - developed at MIT to provide desktop management facilities and as a vehicle for building applications. The system is based upon a client-server model and uses interprocess communication to invoke X Window services. A summary of the X Window system is presented in [Sch86].

Z-buffer - a data structure which contains the z' coordinate of each image space point which has been mapped to screen space. A z-buffer algorithm uses the z' information in the z-buffer to remove hidden-surfaces in the rendered image.

APPENDIX B

APPENDIX B
OCTREE-BASE GRAPHICS OPERATIONS

B.1 INTRODUCTION.

The octree manipulation operations described in this appendix are geometric transformation, hidden-surface removal, and intersection. Because the following discussion relies upon the use of the image overlay algorithm, it is briefly discussed here. The overlay technique uses an overlay octree of resolution 2N to generate a new octree of resolution 2N from an octree of resolution N when reorientation of the original octree can not produce the correct image. A circumstance which illustrates the operation of this procedure is the rotation of the scene by an arbitrary angle. The value of each terminal node in the new octree is determined by finding the value at the intersection of the corresponding terminal node of the original octree with a cell in the overlay . Once the new octree's terminal nodes have been determined, the remainder of the tree is built from the bottom up. The overlay, and therefore the new octree, should have at least twice the resolution as the octree to which the overlay is applied. However, computational costs grow rapidly as resolution increases, so the image created from the new tree reflects a trade-off between speed and accuracy.

B.2 GEOMETRIC TRANSFORMATION.

Rotation by 90° is accomplished by a simple reordering of the nodes of the octree. Rotations of 180° and 270° can be accomplished by reordering as well, except that in these cases the octree is reflected across a plane through the center of the space which is oriented so that the bisecting plane is parallel to an axis of the universe. To accomplish rotation about an arbitrary point, the object is first translated to the center of the universe, rotated, then translated back to its original position. Rotation by an arbitrary angle is accomplished using the overlay technique described above.

Translation uses an octree that represents the object (the old universe), a translation vector, an overlay, and a new octree that represents the translated object (the new universe). The root of the new octree is first offset from the original octree as specified by the translation vector. The overlay is centered on the new octree and restricted in size so that it does not extend past the boundary of the old universe. The two octrees are then traversed in parallel, starting at the root node, so that the translation operation is performed between corresponding nodes of each octree. Node values in the new octree are generated by determining intersection values between the nodes of the old octree and cells in the overlay. The overlay cells determine which sections of the old universe cover the new universe. If a terminal value for a node in the new octree is generated, the descendents of the corresponding node in the original octree need not be examined. The algorithm requires the use of twice as many nodes at a level in the new octree as were used at the corresponding level of the old octree.

Scaling an object by a power of two in all dimensions is accomplished by adding/deleting a root node. An object is halved in all its dimensions by adding a new root node to the tree and adding seven empty siblings to the level of the old root node. A selected portion of the object is doubled in dimension by deleting the old root node and the seven siblings of the node that contains the space to be doubled. Scaling by other than a factor of two requires the use of overlays.

B.3 HIDDEN-SURFACE REMOVAL.

Hidden-surface removal is easily accomplished. The only requirement is that the nodes of the octree be written to the display screen in the proper front-to-back tree traversal sequence, which is determined based on the observer's viewpoint. There is no need to sort the nodes according to z-value as this was accomplished during the octree building procedure. The outcome of the back-to-front procedure is a display with all hidden surfaces removed.

B.4 INTERSECTION DETECTION.

Intersection is used to determine which portions of the object are visible. The input tree is the octree, the output tree is a quadtree. The octree is traversed in back-to-front order, which accomplishes hidden-surface removal. The cubes represented by each of the octree nodes are mapped onto the display screen, with the result of these projections recorded in the quadtree for later display. Because only three sides of each volume are visible from any viewpoint, only three of the six faces of each cube need be examined. The key to the procedure is determining which faces are visible and which are obscured. The visible faces of the cubes in the octree are determined by finding the intersection of the quadtree nodes with the faces of the cubes in the octree. Because it is computationally expensive to accomplish intersection detection directly a different approach is used. Define a bounding box to be the smallest 2D box whose edges are parallel to the quadtree coordinate system which can enclose three faces of the cube associated with a given level of the octree. Intersection detection is performed using the bounding box, the projected node of the octree and the four lowest level nodes of the quadtree. Each of the four lowest level nodes in the quadtree each have a bounding box associated with it. The quadtree nodes lie in the vicinity of the projected octree nodes but are not necessarily co-planar with them.

Intersection detection takes place in two steps. In the first step, the projected octree node is compared to the bounding box associated with each quadtree node. If the result of the comparison is negative, then there is no intersection. If the result is positive, a second test is performed using the six outer edges of the projected node and the area defined by the quadtree node. The lines defined by the projected node's edges are each extended in turn. These lines divide the plane into two parts, the half-plane containing the octree node is called the positive side. If the quadtree box lies entirely on the negative side of any of the lines then the octree cube and the quadtree box do not intersect. If there is an intersection, the quadtree node is labeled appropriately, and the octree node is examined at a lower level if the octree node is not a leaf node. If there is no intersection, then the projected octree node and all its children can be ignored since they must be obscured.

APPENDIX C

APPENDIX C
BASIC RECURSIVE HIDDEN-SURFACE REMOVAL ALGORITHMS

C.1 INTRODUCTION.

This appendix presents the basic back-to-front and front-to-back recursive hidden-surface removal algorithms using the octant numbering scheme presented in Figure 3.2. Depending upon whether the implementation uses a sub-cube center or front-lower left criteria for selecting the $\{\mathcal{P}_0\}$ set, one of two different adjustments to these algorithms must be made at the deepest level of recursion in order to generate correct image space and object space coordinate values. Rather than complicate the presentation of the algorithms with these conditions, they are described here with the understanding that they are engaged when required. First, because for a front-lower left (FLL) $\{\mathcal{P}_O\}$ rotation is best performed by rotating the scene about the origin without translating the scene center to the origin, negative values for image space coordinates, and hence image array indices, are produced at various rotations. To insure that the image is written into the image output buffer, an image array offset equal to N is added to each image array coordinate. The image array is correspondingly expanded to a size of 2N x 2N. For a sub-cube center (SCC) $\{\mathcal{P}_O\}$, an image array index offset is not required. Second, for a SCC $\{\mathcal{P}_O\}$, the algorithm produces the set of coordinates at sub-cube octant 3 and not the sub-cube center at recursion level m-1. This mis-positioning can be corrected by adjusting each of the coordinates generated at level m-1 by applying R twice to $\{\mathcal{V}_{I3}\}$ level m-1 and subtracting the resultant vector from each coordinate set at level m-1. The FLL technique does not have this problem as it is designed to produce coordinate values from FLL octant coordinate values and not sub-cube center values.

C.2 BACK-TO-FRONT ALGORITHM.

Given:

a supersampled scene size of N x N x N where $N = 2^m$
volume center or front-lower left coordinates $\{V_c\} = \{x_c, y_c, z_c\}$
object space octant generation coordinate set $\{O\} = \{\{O_{0_x}, O_{0_y}, O_{0_z}\} \ldots \{O_{7_x}, O_{7_y}, O_{7_z}\}$
map - a function which returns the original object space octant number given an image space octant number
object_array - size N x N x N which contains a 3D image
image_array - size 2N x 2N
Plane_Coeffs - the coefficients of the equation which positions the cutting plane in image space.
behind(I_x, I_y, I_z, Plane_Coeffs) - a function which determines if the given image space coordinates are lying on or behind the cutting plane.

```
until FOREVER do
  {
  acquire_user_display_requirements(x_rot, y_rot, z_rot, Plane_Coeffs)
  /* compute image space volume center coordinates {$V_i$} */
  compute_image_space_coordinates($x_c$,$y_c$,$z_c$, $x'_c$, $y'_c$, $z'_c$, x_rot, y_rot, z_rot)
  /* compute image space octant center coordinate set {I} */
  for s = 0 to 7 do
  compute_image_space_coordinates($O_{s_x}$, $O_{s_y}$, $O_{s_z}$, $I_{s_x}$, $I_{s_y}$, $I_{s_z}$,x_rot, y_rot, z_rot)
  /* sort image space coordinates sets into back to front order, with deepest
  back octant's  coordinates in the leading set position */
  back-to-front_sort({I})
  /* recursively compute all image space coordinates */
  recurse_display(O, I, $x_c$, $y_c$, $z_c$, $x'_c$, $y'_c$, $z'_c$, 1)
  }
```

```
*recurse_display*(**O**, **I**, $C_{0x}$, $C_{0y}$, $C_{0z}$, $C_{ix}$, $C_{iy}$, $C_{iz}$, recursion_level)
{
for j = 0 to 7 do
{
k = *map*(j)
/* compute new image space and object space octant center coordinates */
 if j = (1,3,5,7)  then $C_{x'} = C_{ix} + (I_{jx} >> \text{recursion\_level}^{1})$
           else $C_{x'} = C_{ix} - (I_{jx} >> \text{recursion\_level})$
 if j = (2,3,6,7)  then $C_{y'} = C_{iy} + (I_{jy} >> \text{recursion\_level})$
           else $C_{y'} = C_{iy} - (I_{jy} >> \text{recursion\_level})$
 if j = (0,1,2,3)  then $C_{z'} = C_{iz} + (I_{jz} >> \text{recursion\_level})$
           else $C_{z'} = C_{iz} - (I_{jz} >> \text{recursion\_level})$
 if k = (1,3,5,7)  then $C_x = C_{0x} + (O_{kx} >> \text{recursion\_level})$
           else $C_x = C_{0x} - (O_{kx} >> \text{recursion\_level})$
 if k = (2,3,6,7)  then $C_y = C_{0y} + (O_{ky} >> \text{recursion\_level})$
           else $C_y = C_{0y} - (O_{ky} >> \text{recursion\_level})$
 if k = (0,1,2,3)  then $C_z = C_{0z} + (O_{kz} >> \text{recursion\_level})$
           else $C_z = C_{0z} - (O_{kz} >> \text{recursion\_level})$

 if recursion_level = $\log_2(N)$ and*behind*($C_{x'}$, $C_{y'}$, $C_{z'}$, Plane_Coeffs)
      then image_array[$C_{x'}$][$C_{y'}$] = object_array [$C_x$][$C_y$][$C_z$]
 else *recurse_display*(**O**, **I**, $C_x$, $C_y$, $C_z$, $C_{x'}$, $C_{y'}$, $C_{z'}$, recursion_level+1)

 }}
```

C.3 FRONT-TO-BACK ALGORITHM.

Given:

a supersampled scene size of N x N x N where $N = 2^m$
volume center front-lower left coordinates $\{V_c\} = \{x_c, y_c, z_c\}$
object space octant generation coordinate set $\{O\} = \{\{O_{0x}, O_{0y}, O_{0z}\} \ldots \{O_{7x}, O_{7y}, O_{7z}\}$
map - a function which returns the original object space octant number given an image space octant number

[1]>> is defined to be the right-shift operator.

```
image_array - size 2N x 2N, initialized to -1
object_array - size N x N x N which contains a 3D image
Plane_Coeffs - the coefficients of the equation which positions the cutting
        plane in image space.
behind($I_x$, $I_y$, $I_z$, Plane_Coeffs ) - a function which determines if the given
        image space coordinates are lying on or behind the cutting plane.

until FOREVER do
  {
  acquire_user_display_requirements(x_rot, y_rot, z_rot, Plane_Coeffs)
/* compute image space volume center coordinates {$V_i$} */
  compute_image_space_coordinates($x_c$,$y_c$,$z_c$, $x'_c$, $y'_c$, $z'_c$, x_rot, y_rot, z_rot)
/* compute image space octant center coordinate set {I} */
  for s = 0 to 7 do
  compute_image_space_coordinates($O_{s_x}$, $O_{s_y}$, $O_{s_z}$, $I_{s_x}$, $I_{s_y}$, $I_{s_z}$,x_rot, y_rot, z_rot)
/* sort image space coordinates sets into back to front order, with furthest
  forward octant's coordinates in leading set position */
  front_to_back_sort({I})
/* recursively compute all image space coordinates */
  recurse_display(O, I, $x_c$, $y_c$, $z_c$, $x'_c$, $y'_c$, $z'_c$, 1)
  }


recurse_display(O, I, $C_{o_x}$, $C_{o_y}$, $C_{o_z}$, $C_{i_x}$, $C_{i_y}$ , $C_{i_z}$, recursion_level)
  {
  for j = 0 to 7 do
  {
  k = map(j)
/* compute new image space and object space octant center coordinates */
  if j = (1,3,5,7)   then $C_{x'} = C_{i_x} + (I_{j_x}$ >> recursion_level[1])
             else $C_{x'} = C_{i_x} - (I_{j_x}$ >> recursion_level)
  if j = (2,3,6,7)   then $C_{y'} = C_{i_y} + (I_{j_y}$ >> recursion_level)
             else $C_{y'} = C_{i_y} - (I_{j_y}$ >> recursion_level)
  if j = (0,1,2,3)   then $C_{z'} = C_{i_z} + (I_{j_z}$ >> recursion_level)
             else $C_{z'} = C_{i_z} - (I_{j_z}$ >> recursion_level)
  if k = (1,3,5,7)   then $C_x = C_{o_x} + (O_{k_x}$ >> recursion_level)
             else $C_x = C_{o_x} - (O_{k_x}$ >> recursion_level)
  if k = (2,3,6,7)   then $C_y = C_{o_y} + (O_{k_y}$ >> recursion_level)
             else $C_y = C_{o_y} - (O_{k_y}$ >> recursion_level)
  if k = (0,1,2,3)   then $C_z = C_{o_z} + (O_{k_z}$ >> recursion_level)
             else $C_z = C_{o_z} - (O_{k_z}$ >> recursion_level)
```

[1]>> is defined to be the right-shift operator.

```
if recursion_level = $\log_2(N)$ and image_array[$C_{x'}$][$C_{y'}$] = -1
                    and*behind*($C_{x'}$, $C_{y'}$, $C_{z'}$, Plane_Coeffs)
    then image_array[$C_{x'}$][$C_{y'}$] = object_array [$C_x$][$C_y$][$C_z$]
else *recurse_display*(**O**, **I**, $C_x$, $C_y$, $C_z$, $C_{x'}$, $C_{y'}$, $C_{z'}$, recursion_level+1)
}}
```

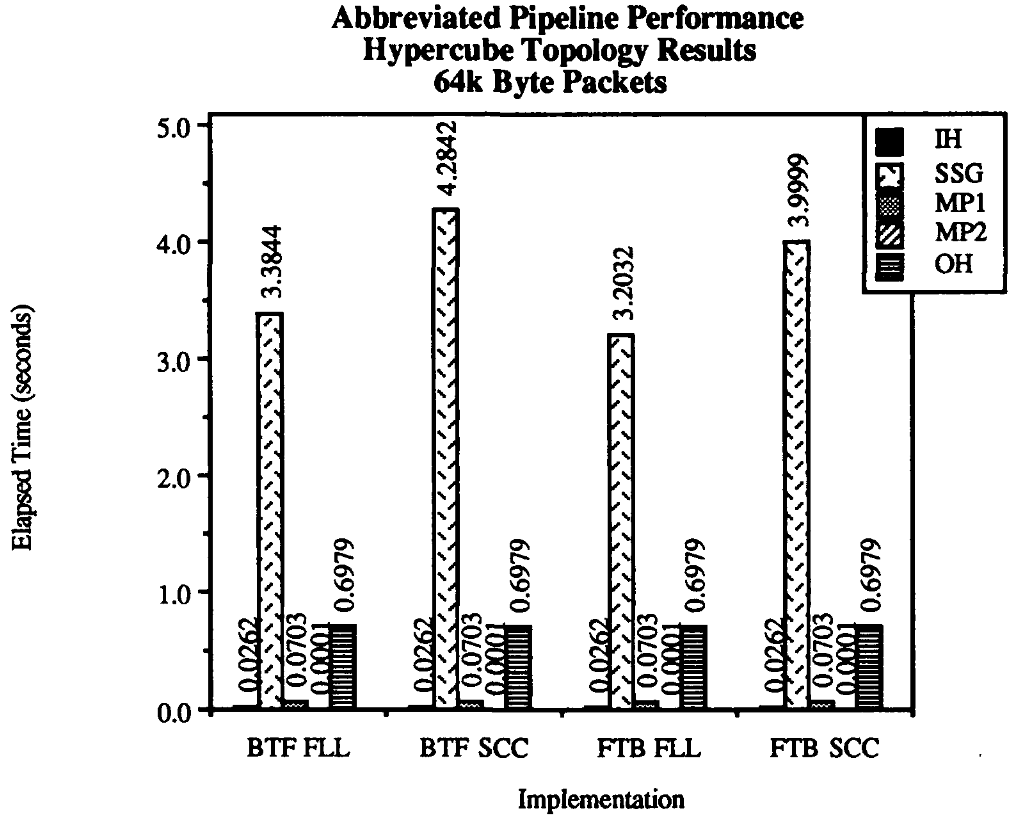
APPENDIX D

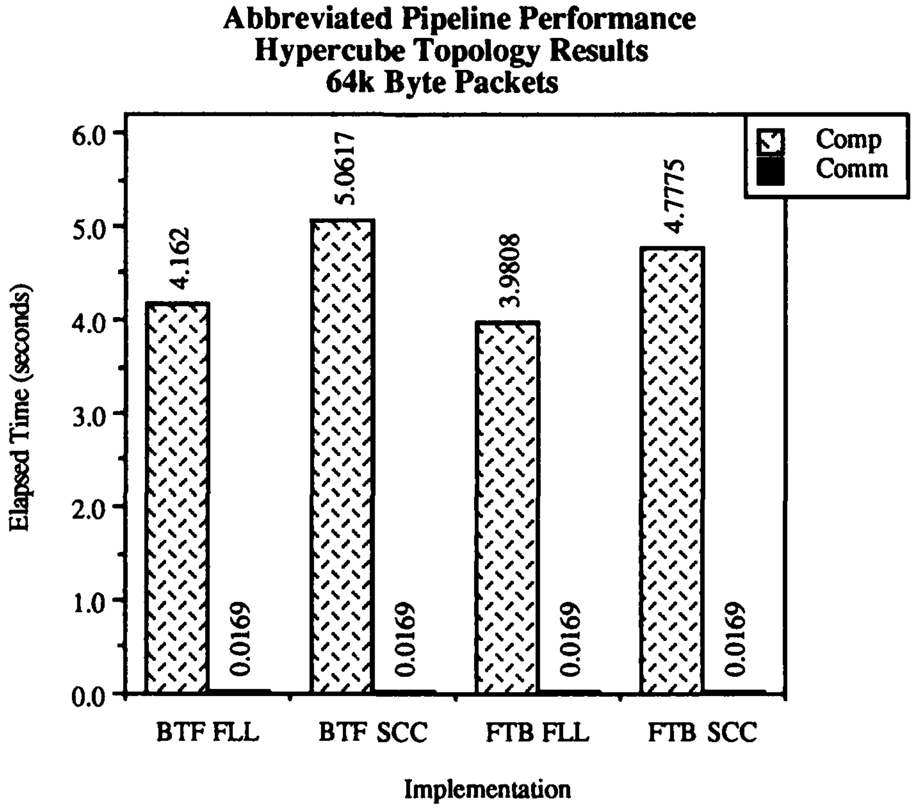
APPENDIX D
INITIAL PIPELINE PERFORMANCE RESULTS

D.1 INTRODUCTION.

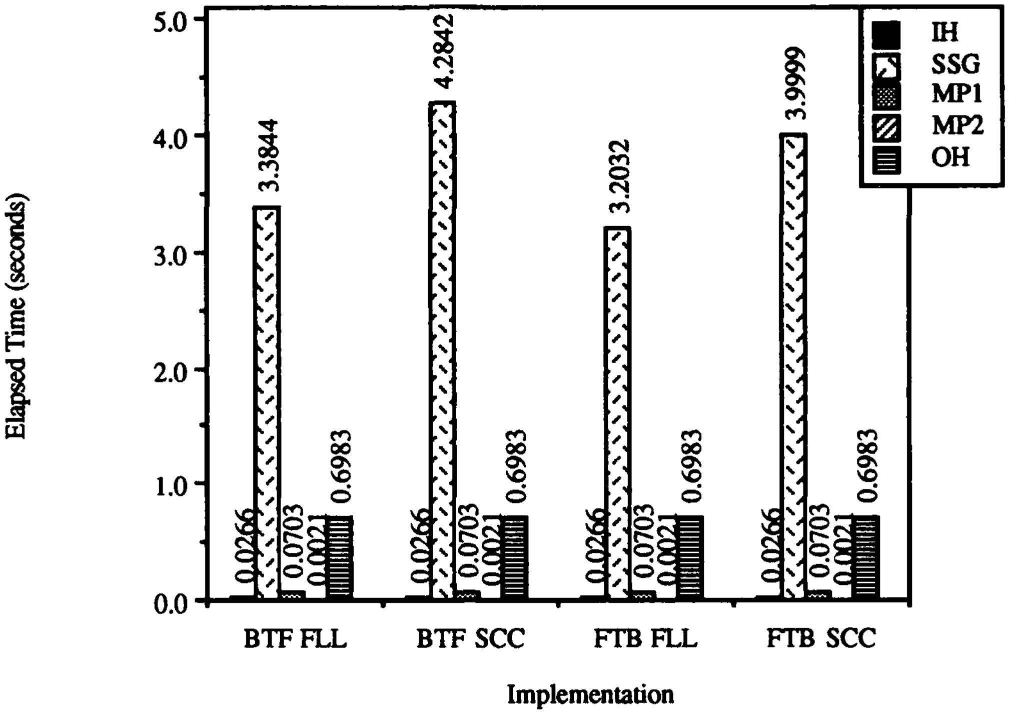
The performance results presented in this appendix depict the performance of the stages of the abbreviated image processing pipeline described in Chapter V. The performance is examined from two aspects. For each testbed machine's interconnection topology (mesh and hypercube) four packet sizes were tested, 512 byte, 4k byte, 16k byte, and 64k byte. For each packet size, the performance of basic back-to-front (BTF) and front-to-back (FTB) hidden-surface removal (HSR) algorithms using either front-lower left coordinates (FLL) or subscene center coordinates (SCC) in the Sub-scene Generator (SSG) stage were tested. The results for each combination of testbed machine topology and packet size are presented in two graphs which contain performance data for both types of BTF and FTB HSR algorithms. The first graph in each sequence illustrates the performance of each stage of the pipeline as the sum of each stage's communication and computation elapsed time. The second graph presents the total computation time vs total communication time at a given packet size for all four types of HSR algorithms. Hypercube topology results for the 4 packet sizes in decreasing packet size order are provided first, followed by mesh topology results.

D.2. HYPERCUBE TOPOLOGY RESULTS.

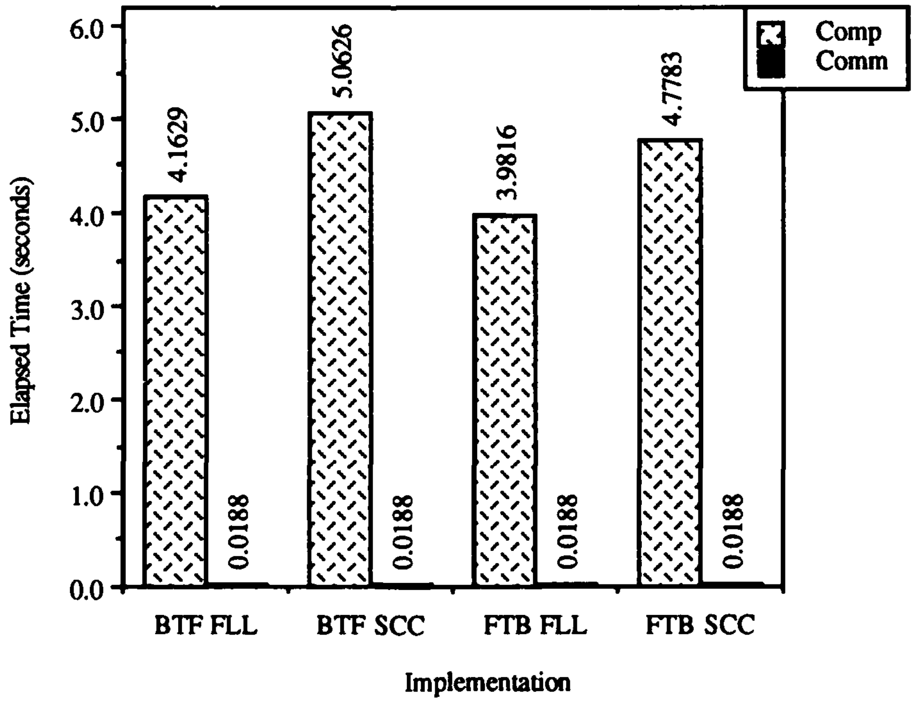




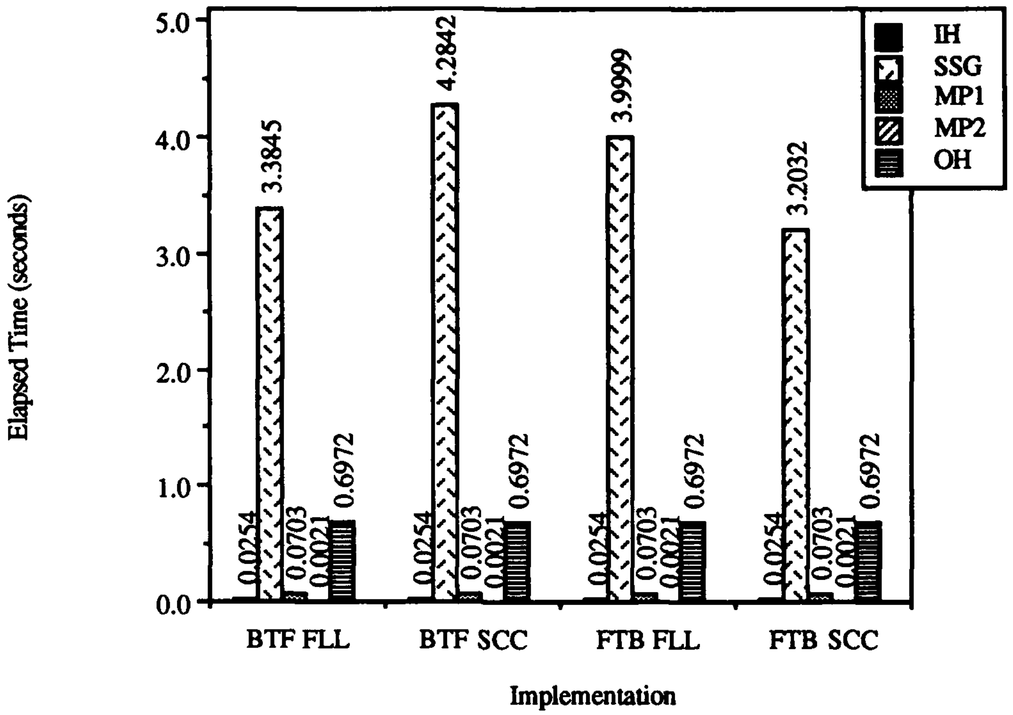
Abbreviated Pipeline Performance
Hypercube Topology Results
16k Byte Packets



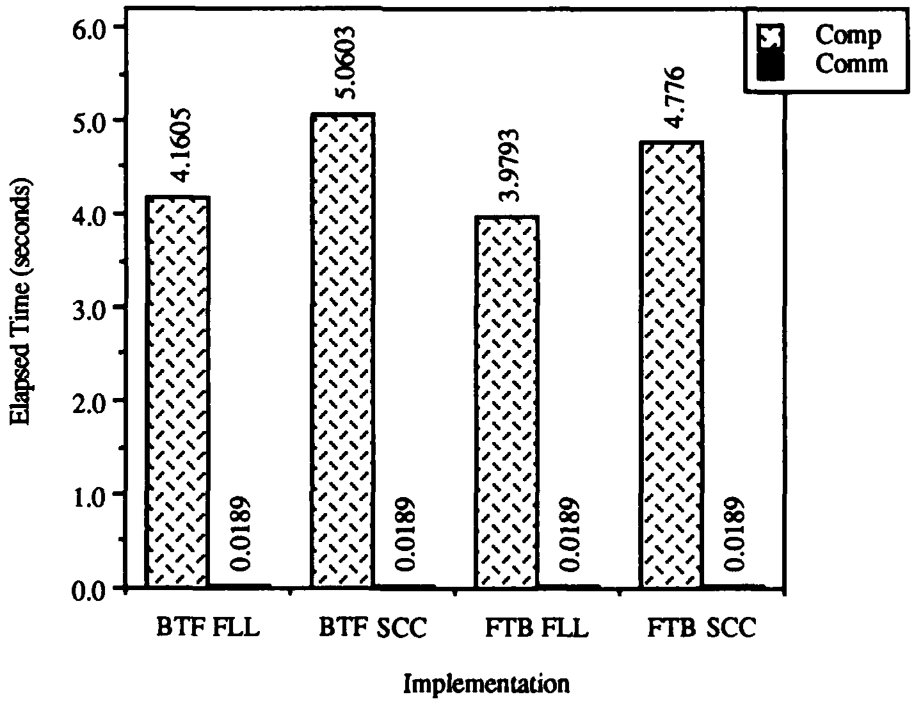
Abbreviated Pipeline Performance
Hypercube Topology Results
16k Byte Packets



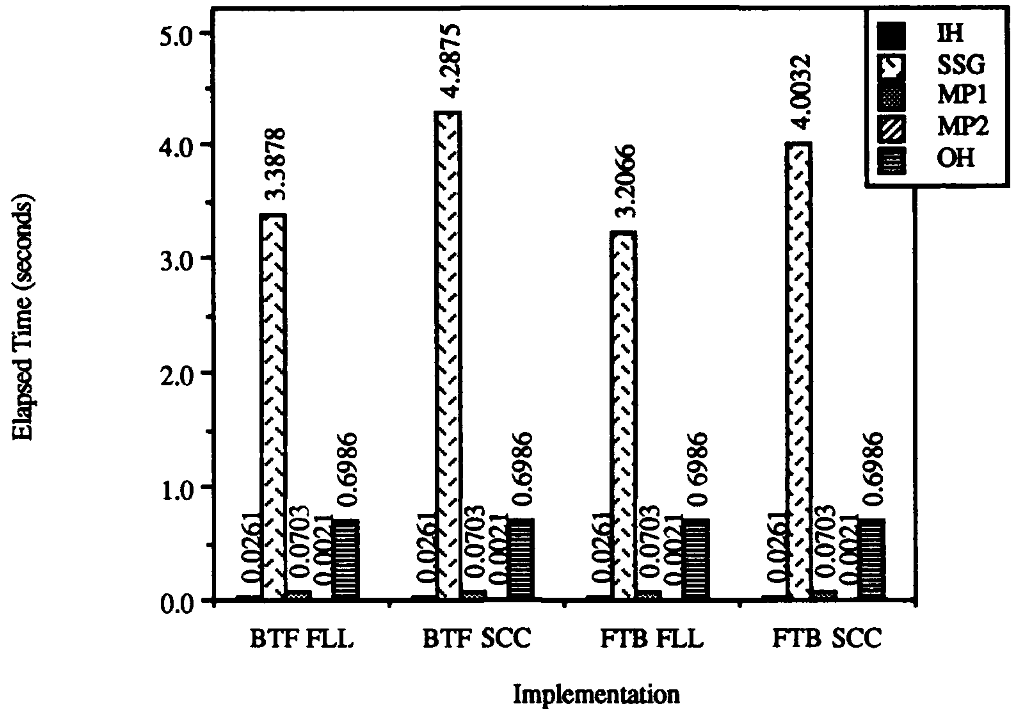
**Abbreviated Pipeline Performance
Hypercube Topology Results
4k Byte Packets**



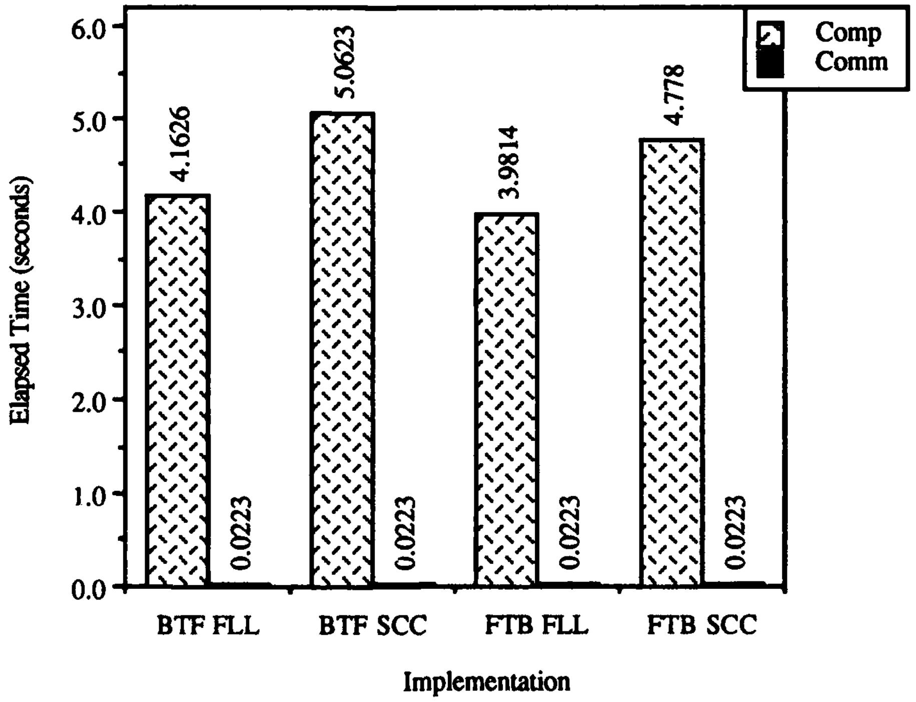
**Abbreviated Pipeline Performance
Hypercube Topology Results
4k Byte Packets**



**Abbreviated Pipeline Performance
Hypercube Topology Results
512 Byte Packets**

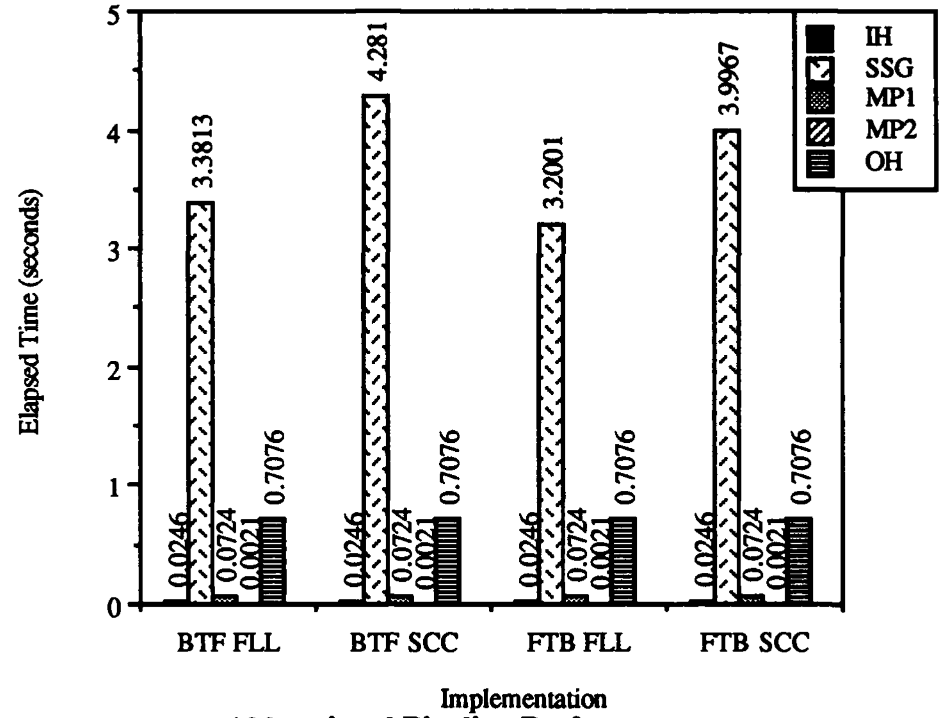


**Abbreviated Pipeline Performance
Hypercube Topology Results
512 Byte Packets**

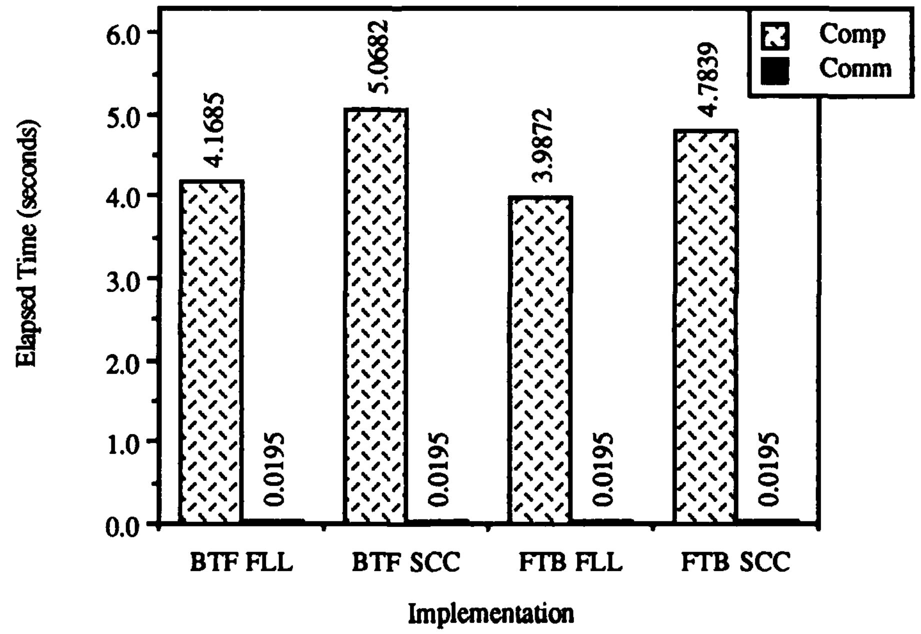


D.3. MESH TOPOLOGY RESULTS.

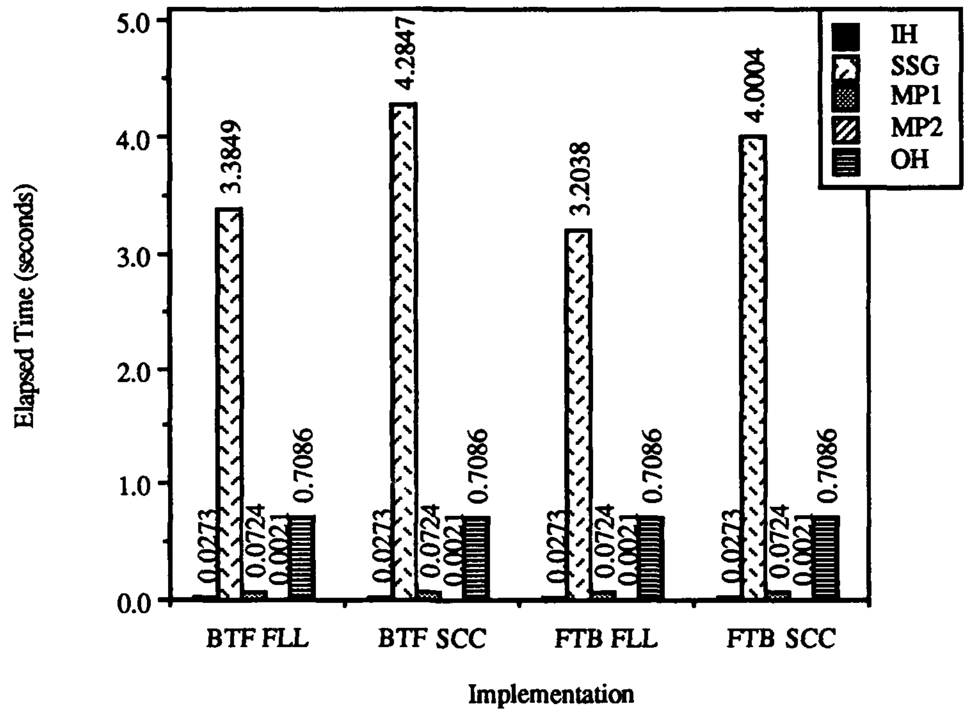
Abbreviated Pipeline Performance
Mesh Topology Results
64k Byte Packets



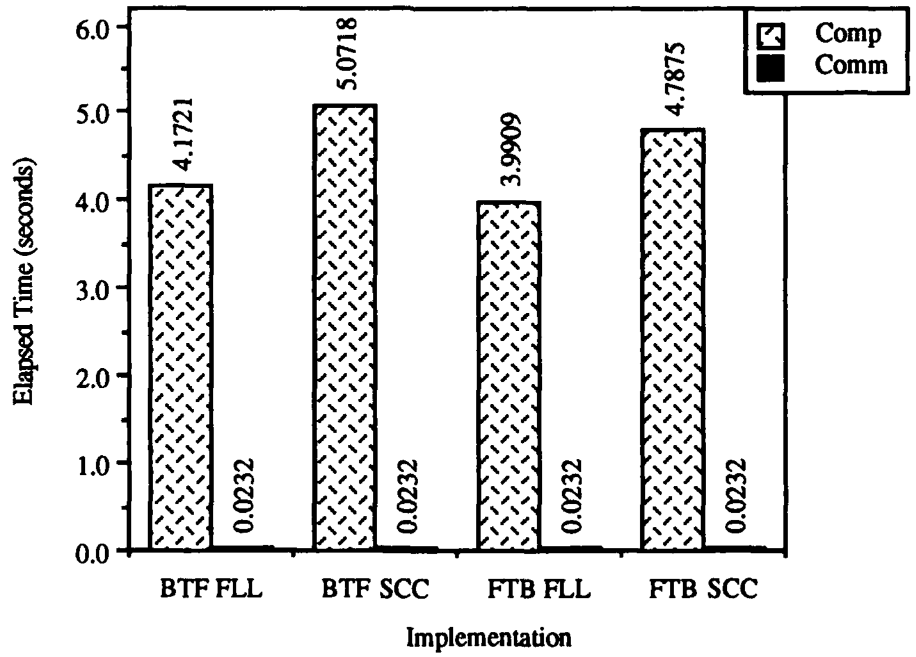
Abbreviated Pipeline Performance
Mesh Topology Results
64k Byte Packets

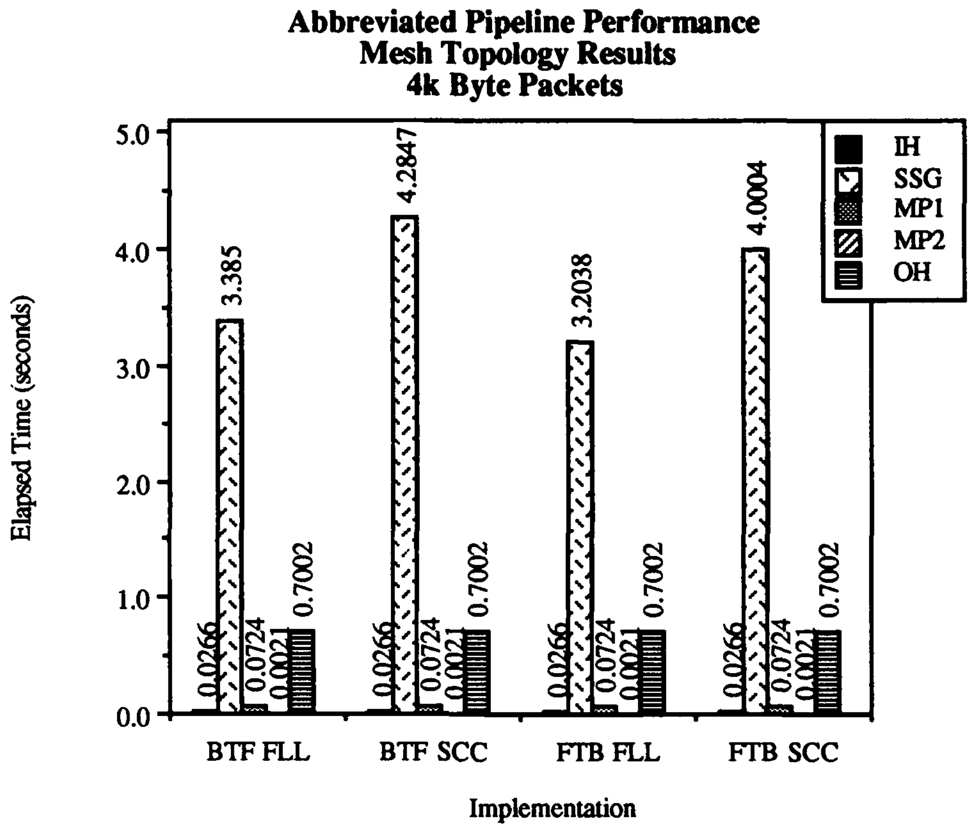


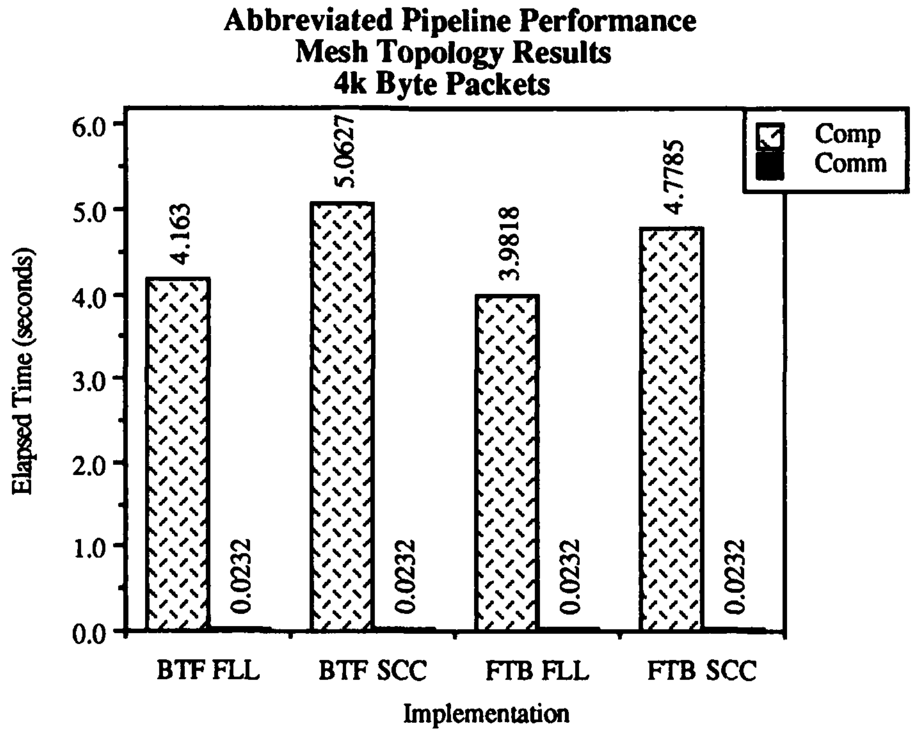
**Abbreviated Pipeline Performance
Mesh Topology Results
16k Byte Packets**



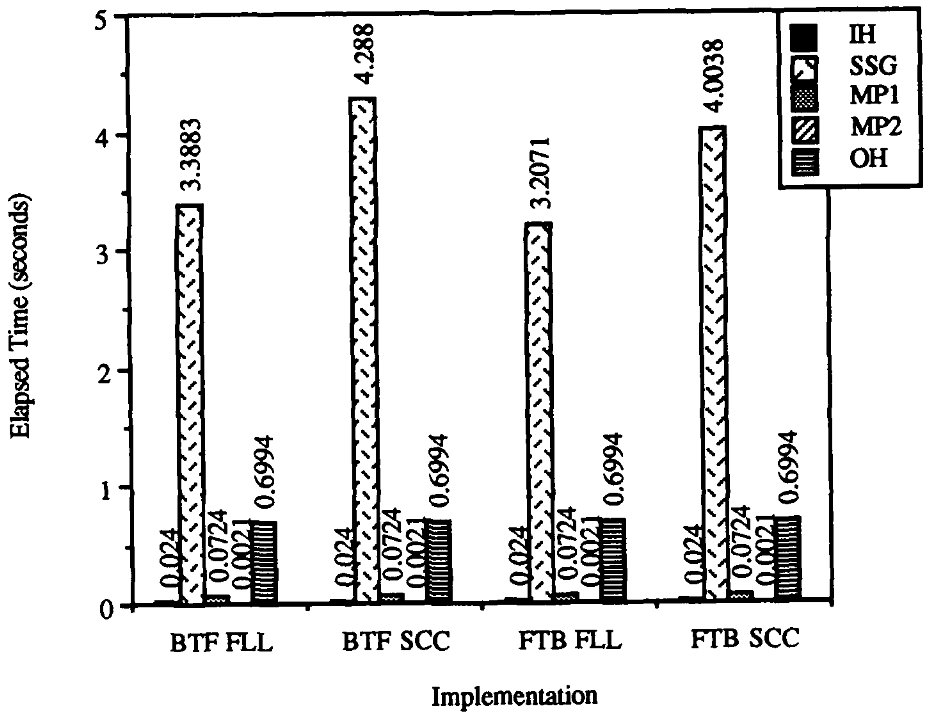
**Abbreviated Pipeline Performance
Mesh Topology Results
16k Byte Packets**



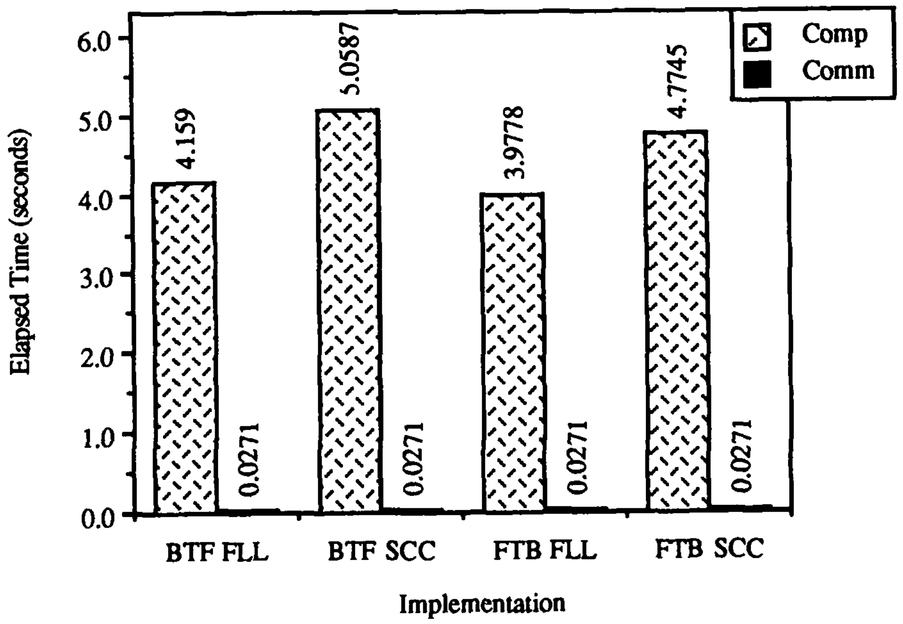

Abbreviated Pipeline Performance
Mesh Topology Results
4k Byte Packets
Elapsed Time (seconds)
5.0
4.0
3.0
2.0
1.0
0.0
3.385
4.2847
3.2038
4.0004
0.0266
0.0724
0.0021
0.7002
BTF FLL
BTF SCC
FTB FLL
FTB SCC
Implementation
IH
SSG
MP1
MP2
OH


Abbreviated Pipeline Performance
Mesh Topology Results
4k Byte Packets
Elapsed Time (seconds)
6.0
5.0
4.0
3.0
2.0
1.0
0.0
4.163
5.0627
3.9818
4.7785
0.0232
BTF FLL
BTF SCC
FTB FLL
FTB SCC
Implementation
Comp
Comm

Abbreviated Pipeline Performance
Mesh Topology Results
512 Byte Packets



Abbreviated Pipeline Performance
Mesh Topology Results
512 Byte Packets



APPENDIX E

APPENDIX E
THE SCENE EDITING RECURSIVE HIDDEN-SURFACE REMOVAL ALGORITHMS

E.1 INTRODUCTION.

This appendix presents the scene editing back-to-front and front-to-back recursive hidden-surface removal algorithms using the octant numbering scheme presented in Figure 3.2. Depending upon whether the implementation uses a sub-cube center or front-lower left criteria for selecting the $\{\mathcal{P}_O\}$ set, one of two different adjustments to these algorithms must be made at the deepest level of recursion in order to generate correct image space and object space coordinate values. Rather than complicate the presentation of the algorithms with these conditions, they are described here with the understanding that they are engaged when required. First, because for a front-lower left (FLL) $\{\mathcal{P}_O\}$ rotation is best performed by rotating the scene about the origin without translating the scene center to the origin, negative values for image space coordinates, and hence image array indices, are produced at various rotations. To insure that the image is written into the image output buffer, an image array offset equal to N is added to each image array coordinate. The image array is correspondingly expanded to a size of 2N x 2N. For a sub-cube center (SCC) $\{\mathcal{P}_O\}$, an image array index offset is not required. Second, for a SCC $\{\mathcal{P}_O\}$, the algorithm produces the set of coordinates at sub-cube octant 3 and not the sub-cube center at recursion level m-1. This mis-positioning can be corrected by adjusting each of the coordinates generated at level m-1 by applying R twice to $\{\mathcal{V}_{I_3}\}$ level m-1 and subtracting the resultant vector from

each coordinate set at level m-1. The FLL technique does not have this problem as it is designed to produce coordinate values from FLL octant coordinate values and not sub-cube center values.

E.2 BACK-TO-FRONT ALGORITHM.

```
Given:
        a supersampled scene size of N x N x N where $N = 2^m$
        volume center or front-lower left coordinates $\{V_c\} = \{x_c, y_c, z_c\}$
        object space octant generation coordinate set $\{O\} = \{\{O_{0_x}, O_{0_y}, O_{0_z}\} \ldots$
                $\{O_{7_x}, O_{7_y}, O_{7_z}\}$
        *map* - a function which returns the original object space octant number given
                an image space octant number
        object_array - size N x N x N which contains a 3D image
        image_array - size 2N x 2N
        Plane_Coeffs - the coefficients of the equation which positions the cutting
                plane in image space.
        *behind*($I_x, I_y, I_z$, Plane_Coeffs ) - a function which determines if the given
                image space coordinates are lying on or behind the cutting plane.
        $BTFRP_O$ - the object space coordinates for the location of the Back-to-Front
                Reference Point, initially positioned at coordinates N/2, N/2, 2N
        *Plane_Distance*($I_x, I_y, I_z$, Plane_Coeffs ) - a function which returns the
                distance from the cutting plane to image space coordinates $I_x, I_y, I_z$
        *BTF_Distance*($I_x, I_y, I_z, BTFRP_I$) - a function which returns the distance
                between the BTFRP and image space coordinates $I_x, I_y, I_z$
        L - length of the x-axis projection of the vector running from an octant's
                generation point to the furthest point in the octant, equal to N/2 for
                the FLL approach and N/4 for the SCC approach

until FOREVER do
  {
  *acquire_user_display_requirements*(x_rot, y_rot, z_rot, Plane_Coeffs)
/* compute image space volume center coordinates $\{V_i\}$ */
  *compute_image_space_coordinates*($x_c, y_c, z_c, x'_c, y'_c, z'_c$, x_rot, y_rot, z_rot)
/* determine image space location of BTFRP */
  *compute_image_space_coordinates*($BTFRP_O, BTFRP_I$, x_rot, y_rot, z_rot)
/* compute image space octant center coordinate set {I} */
  for s = 0 to 7 do
  *compute_image_space_coordinates*($O_{s_x}, O_{s_y}, O_{s_z}, I_{s_x}, I_{s_y}, I_{s_z}$,x_rot, y_rot, z_rot)
/* sort image space coordinates sets into back to front order, with deepest
   octant's coordinates in leading set position */
  *back-to-front_sort*({I})
/* recursively compute all image space coordinates */
recurse_display(O, I, $x_c$, $y_c$, $z_c$, $x'_c$, $y'_c$, $z'_c$, 1)
}

recurse_display(O, I, $C_{o_x}$, $C_{o_y}$, $C_{o_z}$, $C_{i_x}$, $C_{i_y}$, $C_{i_z}$, recursion_level)
{
for j = 0 to 7 do
{
k = map(j)
/* compute new image space and object space octant center coordinates */
  if j = (1,3,5,7)  then $C_x' = C_{i_x} + (I_{j_x}$ >> recursion_level[1])
            else $C_x' = C_{i_x} - (I_{j_x}$ >> recursion_level)
  if j = (2,3,6,7)  then $C_y' = C_{i_y} + (I_{j_y}$ >> recursion_level)
            else $C_y' = C_{i_y} - (I_{j_y}$ >> recursion_level)
  if j = (0,1,2,3)  then $C_z' = C_{i_z} + (I_{j_z}$ >> recursion_level)
            else $C_z' = C_{i_z} - (I_{j_z}$ >> recursion_level)
  if k = (1,3,5,7)  then $C_x = C_{o_x} + (O_{k_x}$ >> recursion_level)
            else $C_x = C_{o_x} - (O_{k_x}$ >> recursion_level)
  if k = (2,3,6,7)  then $C_y = C_{o_y} + (O_{k_y}$ >> recursion_level)
            else $C_y = C_{o_y} - (O_{k_y}$ >> recursion_level)
  if k = (0,1,2,3)  then $C_z = C_{o_z} + (O_{k_z}$ >> recursion_level)
            else $C_z = C_{o_z} - (O_{k_z}$ >> recursion_level)

/* check to determine if current image space coordinate is further from BTFRP than
        parent octant coordinate or if cutting plane passes through the current
        image space octant */

if recursion_level < $\log_2(N)$ and
  (BTF_Distance($C_x'$,$C_y'$,$C_z'$BTFRP$_I$)>BTF_Distance($C_{i_x}$,$C_{i_y}$,$C_{i_z}$,BTFRP$_I$)) or
  (Plane_Distance($C_x'$,$C_y'$,$C_z'$,Plane_Coeffs)<$\sqrt{(L>>(recursion\_level-1))^2 * 3}$)
  then
    recurse_display(O, I, $C_x$, $C_y$, $C_z$, $C_x'$, $C_y'$, $C_z'$, recursion_level+1)

if recursion level = $\log_2(N)$ andbehind($C_x'$, $C_y'$, $C_z'$, Plane_Coeffs)
  then image_array[$C_{x'}$][$C_{y'}$] = object_array [$C_x$][$C_y$][$C_z$]

  }}
```

[1]>> is defined to be the right-shift operator.

E.3 FRONT-TO-BACK ALGORITHM.

Given:

a supersampled scene size of N x N x N where $N = 2^m$
volume center or front-lower left coordinates $\{V_c\} = \{x_c, y_c, z_c\}$
object space octant generation coordinate set $\{O\} = \{\{O_{0_x}, O_{0_y}, O_{0_z}\} \ldots \{O_{7_x}, O_{7_y}, O_{7_z}\}$
map - a function which returns the original object space octant number given an image space octant number
object_array - size N x N x N which contains a 3D image
image_array - size 2N x 2N, initialized to -1
Plane_Coeffs - the coefficients of the equation which positions the cutting plane in image space.
behind(I_x, I_y, I_z, Plane_Coeffs) - a function which determines if the given image space coordinates are lying on or behind the cutting plane.
$FTBRP_O$ - the object space coordinates for the location of the Front-to-Back Reference Point, initially positioned at coordinates N/2, N/2, -2N
Plane_Distance(I_x, I_y, I_z, Plane_Coeffs) - a function which returns the distance from the cutting plane to image space coordinates I_x, I_y, I_z
FTB_Distance(I_x, I_y, I_z, $FTBRP_I$) - a function which returns the distance between the FTBRP and image space coordinates I_x, I_y, I_z
L - length of the x-axis projection of the vector running from an octant's generation point to the furthest point in the octant, equal to N/2 for the FLL approach and N/4 for the SCC approach

```
until FOREVER do
  {
  acquire_user_display_requirements(x_rot, y_rot, z_rot, Plane_Coeffs)
/* compute image space volume center coordinates {V_i} */
  compute_image_space_coordinates($x_c$,$y_c$,$z_c$, $x'_c$, $y'_c$, $z'_c$, x_rot, y_rot, z_rot)
/* determine image space location of FTBRP */
  compute_image_space_coordinates($FTBRP_O$, $FTBRP_I$, x_rot, y_rot, z_rot)
/* compute image space octant center coordinate set {I} */
  for s = 0 to 7 do
  compute_image_space_coordinates($O_{s_x}$, $O_{s_y}$, $O_{s_z}$, $I_{s_x}$, $I_{s_y}$, $I_{s_z}$,x_rot, y_rot, z_rot)
/* sort image space coordinates sets into back to front order, with furthest
  forward octant's coordinates in leading set position */
  front_to_back_sort({I})
/* recursively compute all image space coordinates */
  recurse_display(O, I, $x_c$, $y_c$, $z_c$, $x'_c$, $y'_c$, $z'_c$, 1)
  }
```

```
*recurse_display*(O, I, $C_{o_x}$, $C_{o_y}$, $C_{o_z}$, $C_{i_x}$, $C_{i_y}$, $C_{i_z}$, recursion_level)
  {
  for j = 0 to 7 do
  {
  k = *map*(j)
/* compute new image space and object space octant center coordinates */
  if j = (1,3,5,7)   then $C_{x'} = C_{i_x} + (I_{j_x}$ >> recursion_level[1])
          else $C_{x'} = C_{i_x} - (I_{j_x}$ >> recursion_level)
  if j = (2,3,6,7)   then $C_{y'} = C_{i_y} + (I_{j_y}$ >> recursion_level)
          else $C_{y'} = C_{i_y} - (I_{j_y}$ >> recursion_level)
  if j = (0,1,2,3)   then $C_{z'} = C_{i_z} + (I_{j_z}$ >> recursion_level)
          else $C_{z'} = C_{i_z} - (I_{j_z}$ >> recursion_level)
  if k = (1,3,5,7)   then $C_x = C_{o_x} + (O_{k_x}$ >> recursion_level)
          else $C_x = C_{o_x} - (O_{k_x}$ >> recursion_level)
  if k = (2,3,6,7)   then $C_y = C_{o_y} + (O_{k_y}$ >> recursion_level)
          else $C_y = C_{o_y} - (O_{k_y}$ >> recursion_level)
  if k = (0,1,2,3)   then $C_z = C_{o_z} + (O_{k_z}$ >> recursion_level)
          else $C_z = C_{o_z} - (O_{k_z}$ >> recursion_level)

/* check to determine if current image space coordinate is further from $FTBRP_I$
        than parent octant coordinate or if cutting plane passes through the current
        image space octant */

if recursion_level < $\log_2(N)$ and
  (*FTB_Distance*($C_{x'}$,$C_{y'}$,$C_{z'}$,$FTBRP_I$)<*FTB_Distance*($C_{i_x}$,$C_{i_y}$,$C_{i_z}$,$FTBRP_I$)) or
  (Plane_Distance($C_{x'}$,$C_{y'}$,$C_{z'}$,Plane_Coeffs)<$\sqrt{(L>>(\text{recursion\_level}-1))^2 * 3}$)
  then
    *recurse_display*(O, I, $C_x$, $C_y$, $C_z$, $C_{x'}$, $C_{y'}$, $C_{z'}$, recursion_level+1)

if recursion_level = $\log_2(N)$ and image_array[$C_{x'}$][$C_{y'}$] = -1 and
        *behind*($C_{x'}$, $C_{y'}$, $C_{z'}$, Plane_Coeffs)
  then image_array[$C_{x'}$][$C_{y'}$] = object_array [$C_x$][$C_y$][$C_z$]

  }}
```

[1]>> is defined to be the right-shift operator.

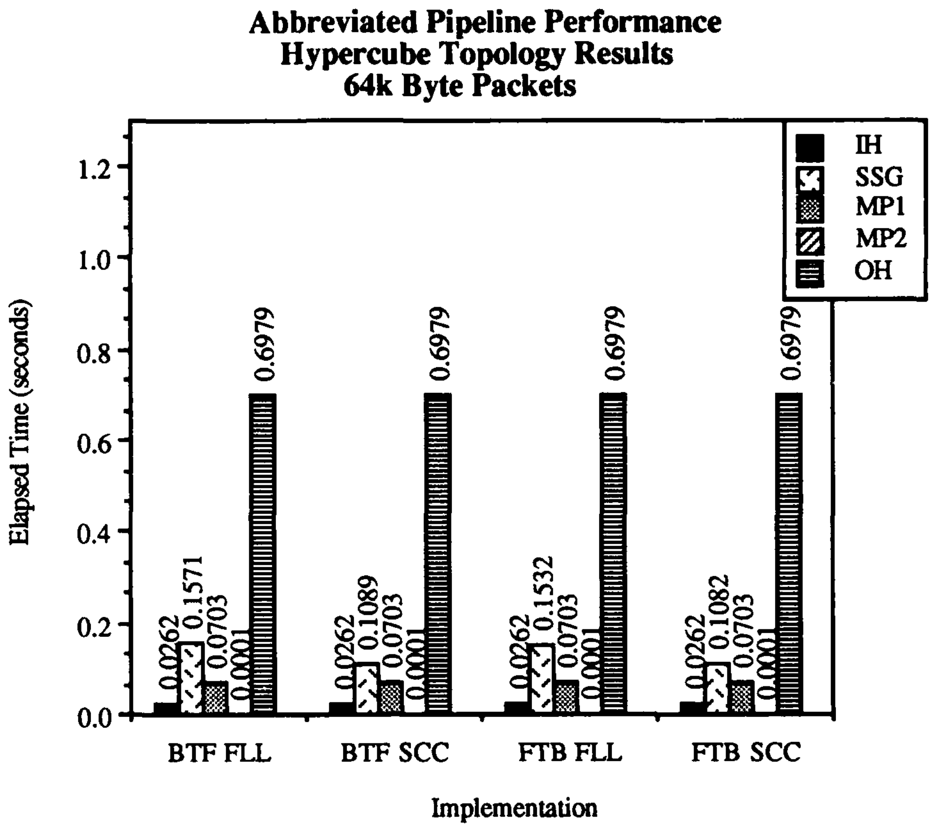
APPENDIX F

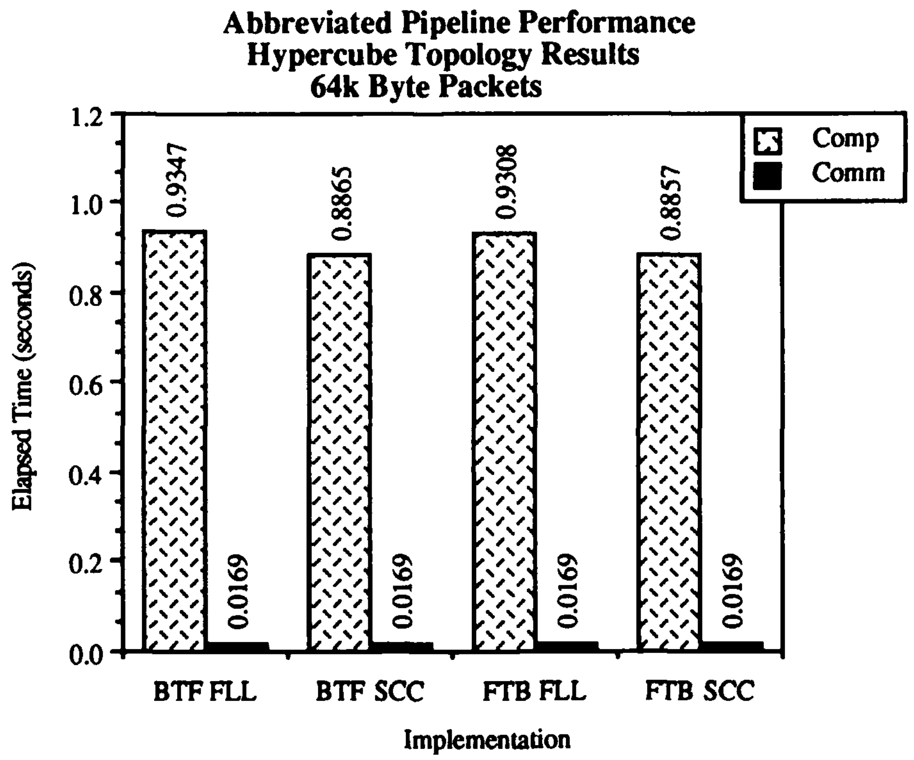
APPENDIX F
PIPELINE PERFORMANCE USING THE SCENE EDITING ALGORITHMS

F.1 INTRODUCTION.

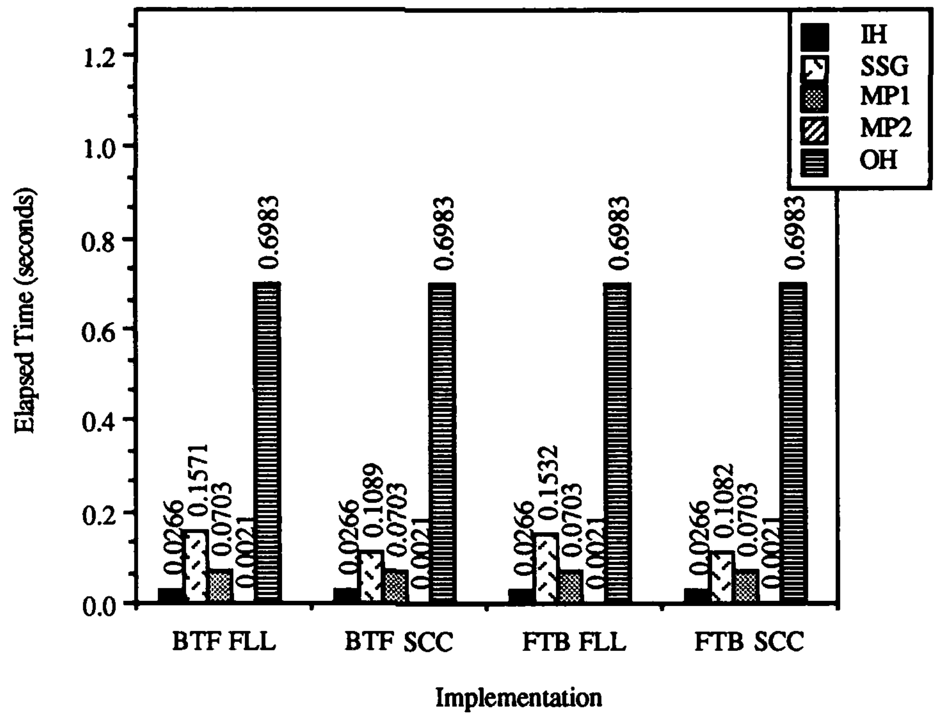
The performance results presented in this appendix depict the performance of the stages of the abbreviated image processing pipeline using the scene editing algorithms described in Chapter V. The performance is examined from two aspects. For each testbed machine's interconnection topology (mesh and hypercube) four packet sizes were tested, 512 byte, 4k byte, 16k byte, and 64k byte. For each packet size, the performance of the scene editing back-to-front (BTF) and front-to-back (FTB) hidden-surface removal (HSR) algorithms using either front-lower left coordinates (FLL) or subscene center coordinates (SCC) in the Sub-scene Generator (SSG) stage were tested. The results for each combination of testbed machine interconnection topology and packet size are presented in a sequence of two graphs which contain performance data for both types of BTF and FTB HSR algorithms. The first graph in each sequence illustrates the performance of each stage of the pipeline as sum of each stage's communication and computation time. The second graph presents the total computation time vs total communication time at a given packet size for all four types of HSR algorithms. Hypercube topology results for the 4 packet sizes in decreasing packet size order are provided first, followed by mesh topology results.

F.2. HYPERCUBE TOPOLOGY RESULTS.

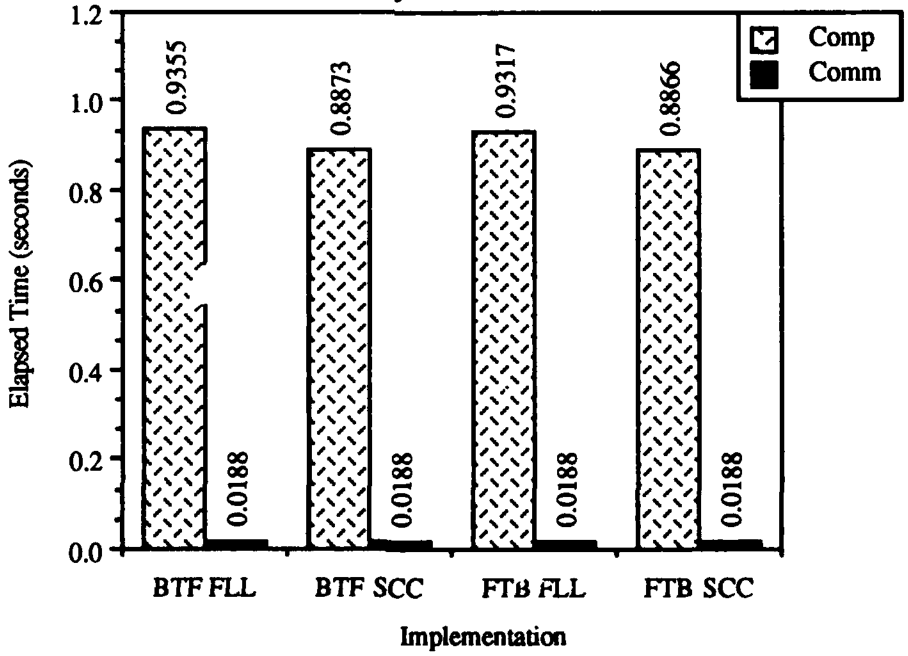


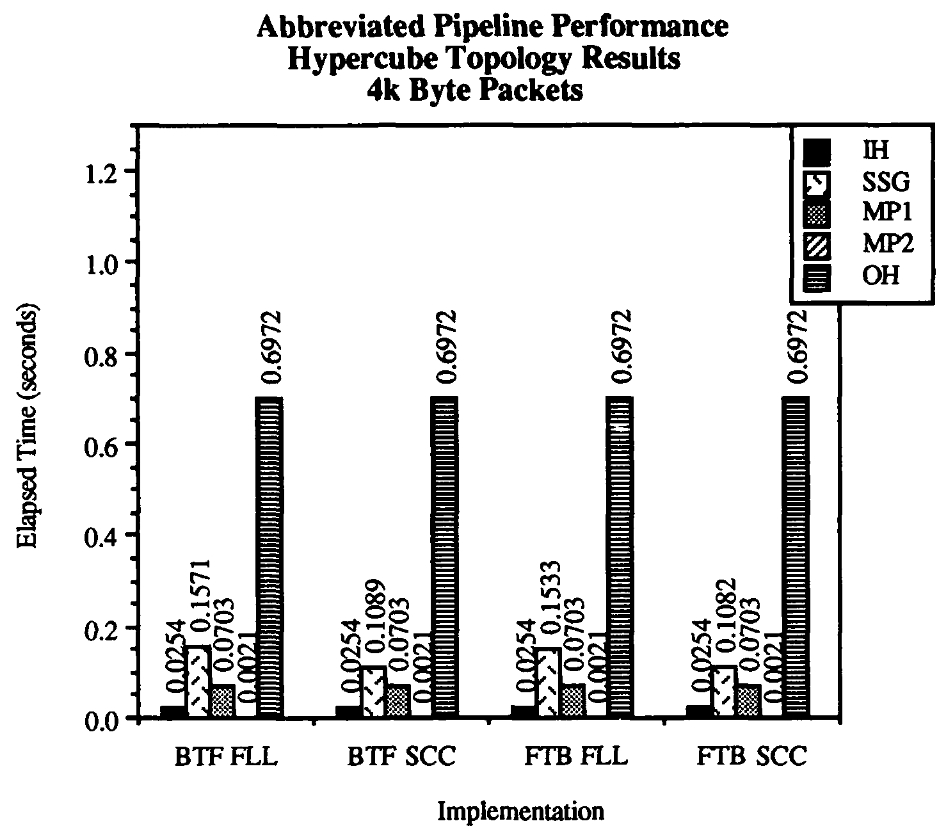


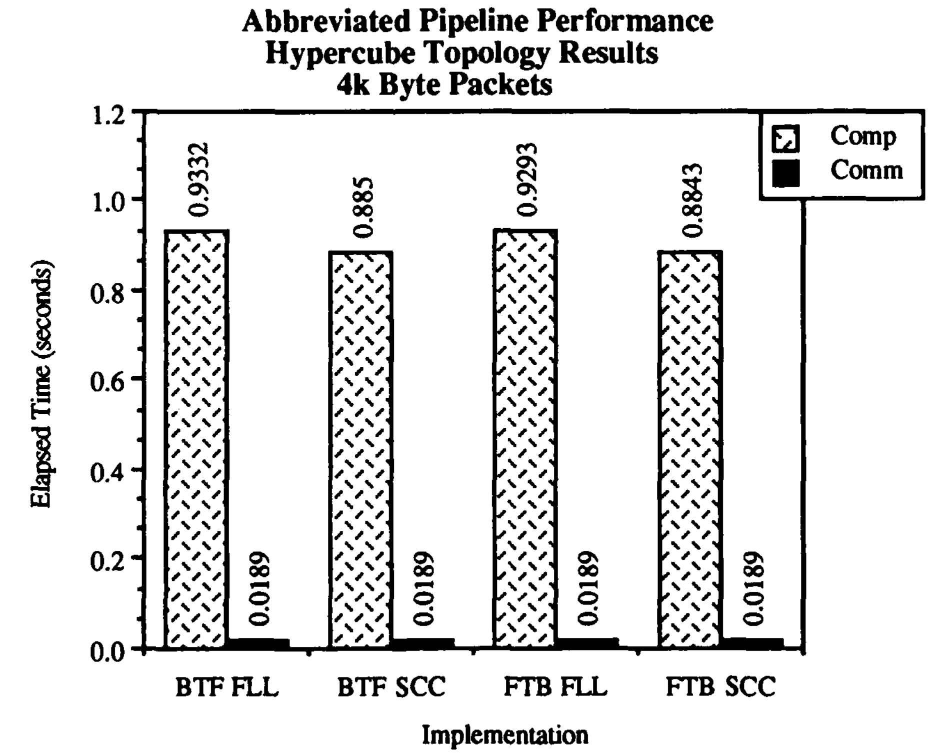
**Abbreviated Pipeline Performance
Hypercube Topology Results
16k Byte Packets**

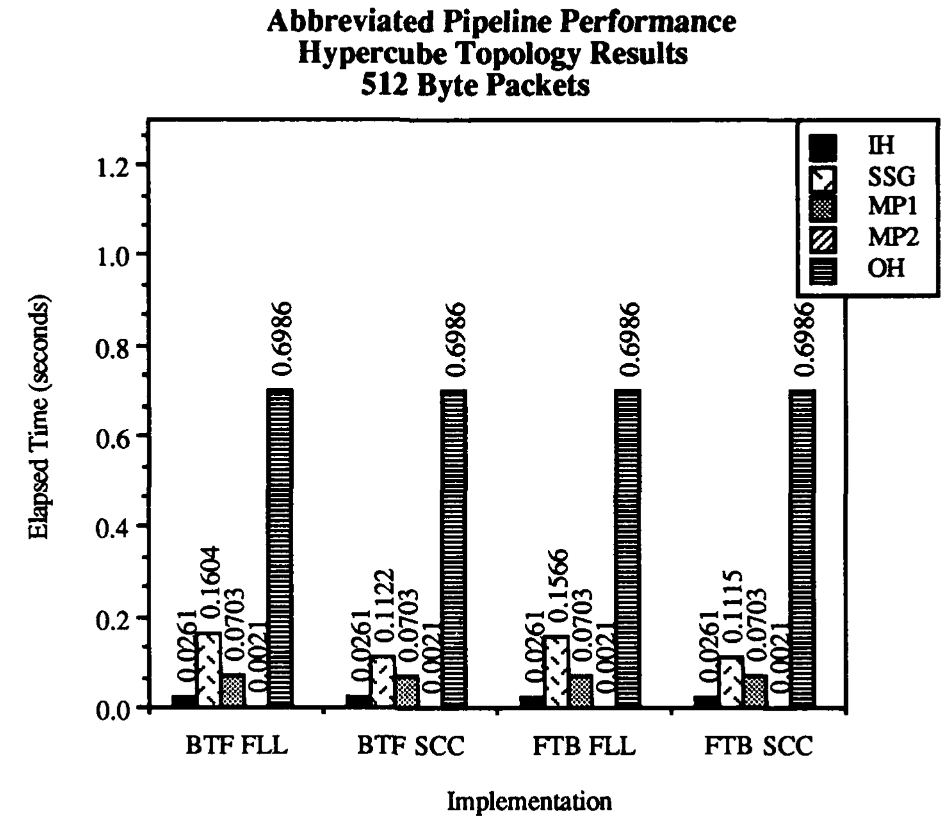


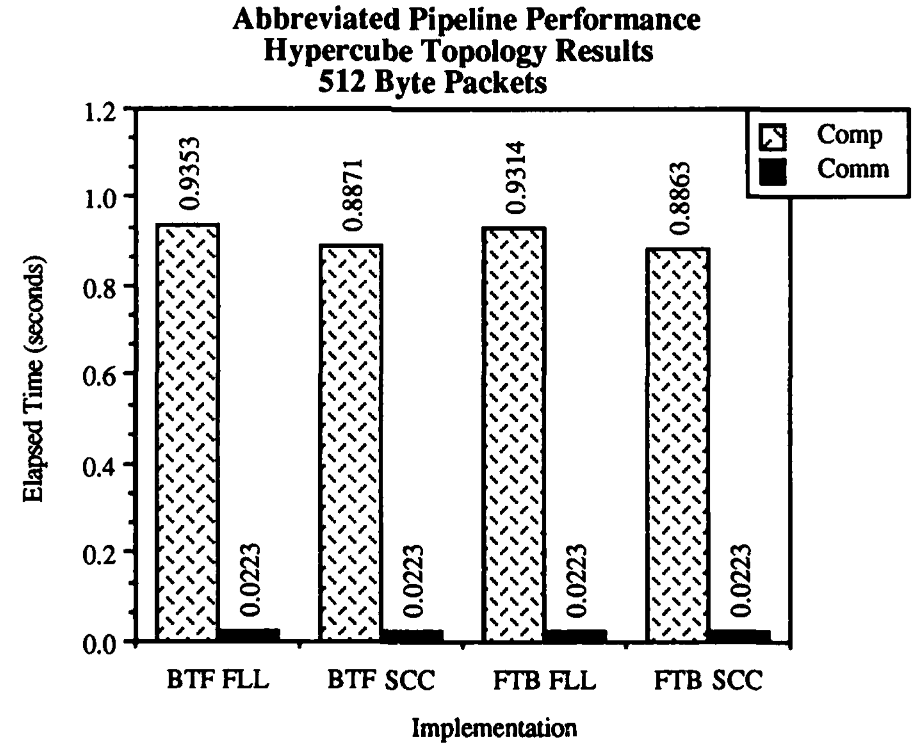
**Abbreviated Pipeline Performance
Hypercube Topology Results
16k Byte Packets**



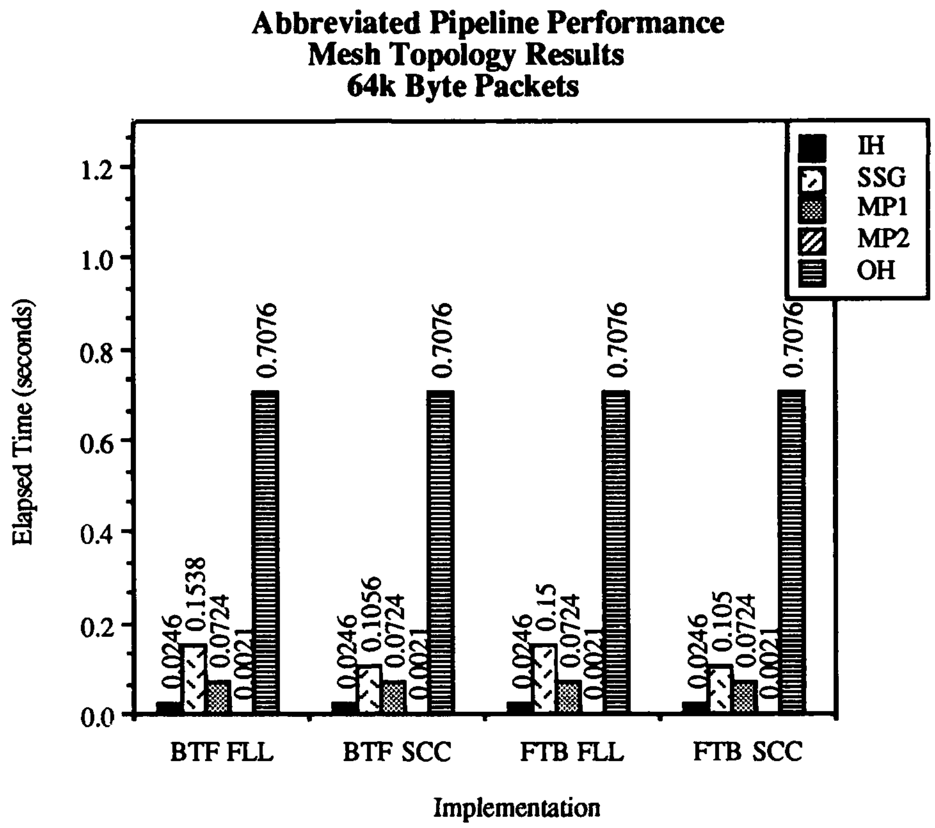

Abbreviated Pipeline Performance
Hypercube Topology Results
4k Byte Packets
Elapsed Time (seconds)
Implementation
IH
SSG
MP1
MP2
OH
BTF FLL
BTF SCC
FTB FLL
FTB SCC
0.0254
0.1571
0.0703
0.0021
0.6972
0.0254
0.1089
0.0703
0.0021
0.6972
0.0254
0.1533
0.0703
0.0021
0.6972
0.0254
0.1082
0.0703
0.0021
0.6972
0.0
0.2
0.4
0.6
0.8
1.0
1.2

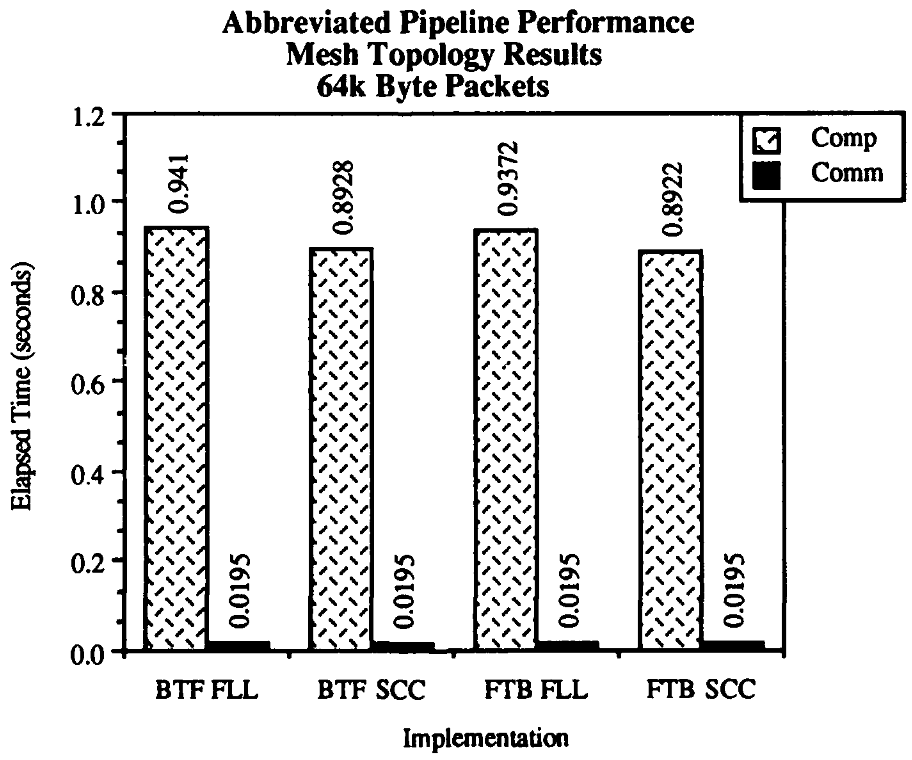

Abbreviated Pipeline Performance
Hypercube Topology Results
4k Byte Packets
Elapsed Time (seconds)
Implementation
Comp
Comm
BTF FLL
BTF SCC
FTB FLL
FTB SCC
0.9332
0.0189
0.885
0.0189
0.9293
0.0189
0.8843
0.0189
0.0
0.2
0.4
0.6
0.8
1.0
1.2

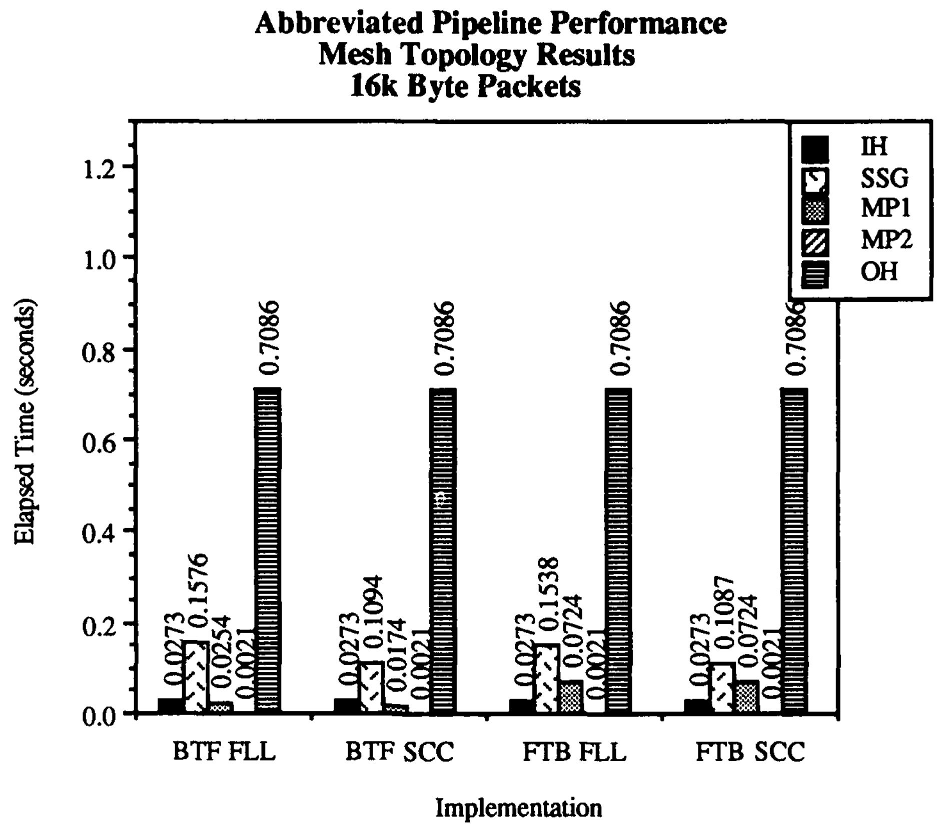

Abbreviated Pipeline Performance
Hypercube Topology Results
512 Byte Packets
Elapsed Time (seconds)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
IH
SSG
MP1
MP2
OH
0.0261
0.1604
0.0703
0.0021
0.6986
0.0261
0.1122
0.0703
0.0021
0.6986
0.0261
0.1566
0.0703
0.0021
0.6986
0.0261
0.1115
0.0703
0.0021
0.6986
BTF FLL
BTF SCC
FTB FLL
FTB SCC
Implementation


Abbreviated Pipeline Performance
Hypercube Topology Results
512 Byte Packets
Elapsed Time (seconds)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Comp
Comm
0.9353
0.0223
0.8871
0.0223
0.9314
0.0223
0.8863
0.0223
BTF FLL
BTF SCC
FTB FLL
FTB SCC
Implementation

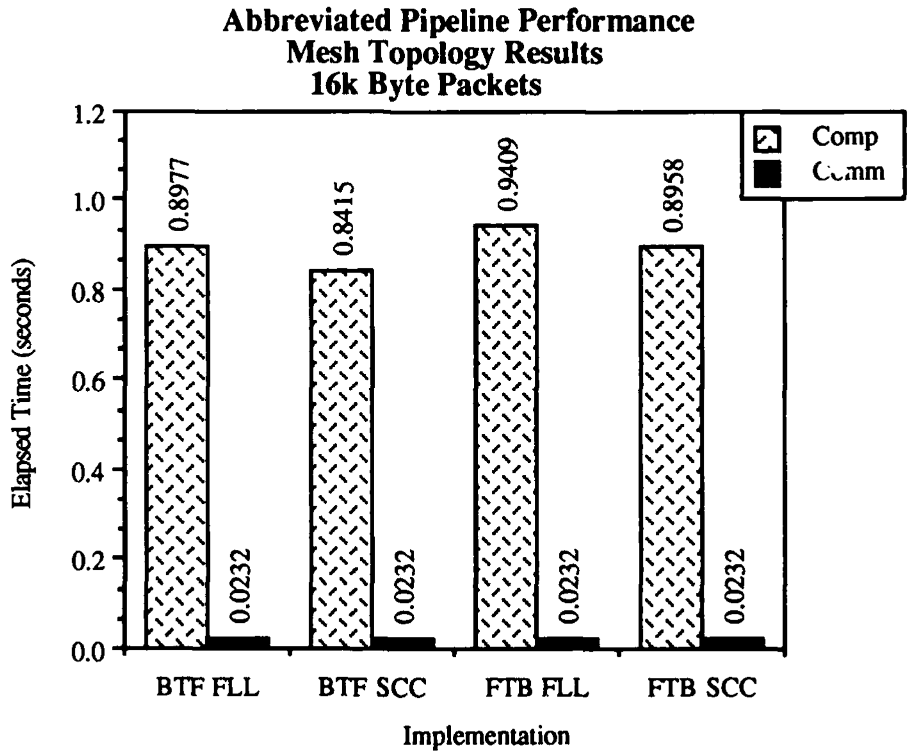
F.3. MESH TOPOLOGY RESULTS.





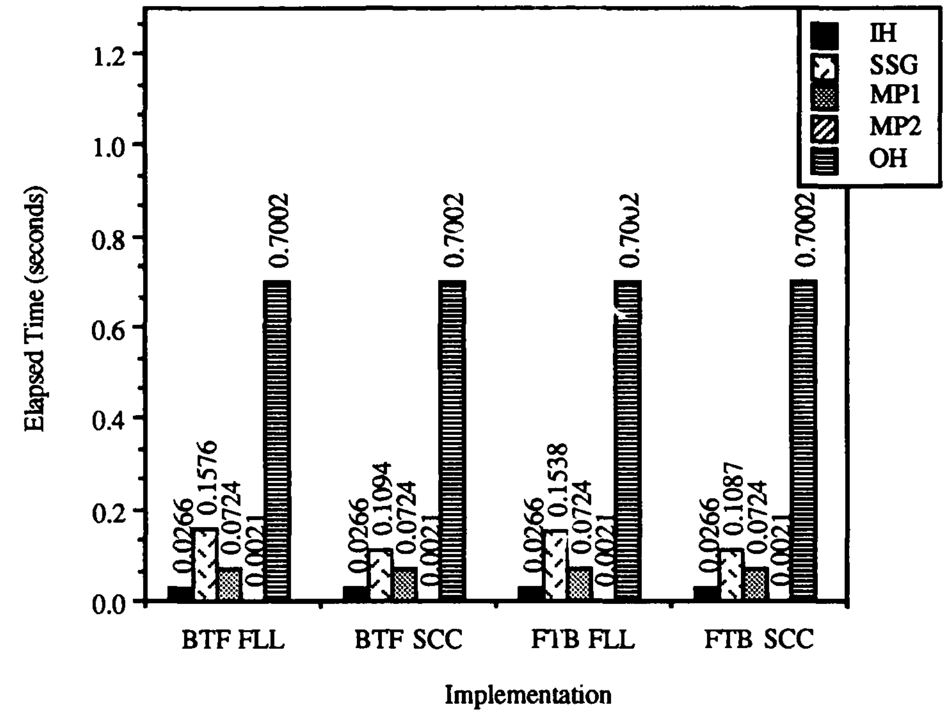


Abbreviated Pipeline Performance
Mesh Topology Results
16k Byte Packets
Elapsed Time (seconds)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
IH
SSG
MP1
MP2
OH
0.0273
0.1576
0.0254
0.0021
0.7086
0.0273
0.1094
0.0174
0.0021
0.7086
0.0273
0.1538
0.0724
0.0021
0.7086
0.0273
0.1087
0.0724
0.0021
0.7086
BTF FLL
BTF SCC
FTB FLL
FTB SCC
Implementation

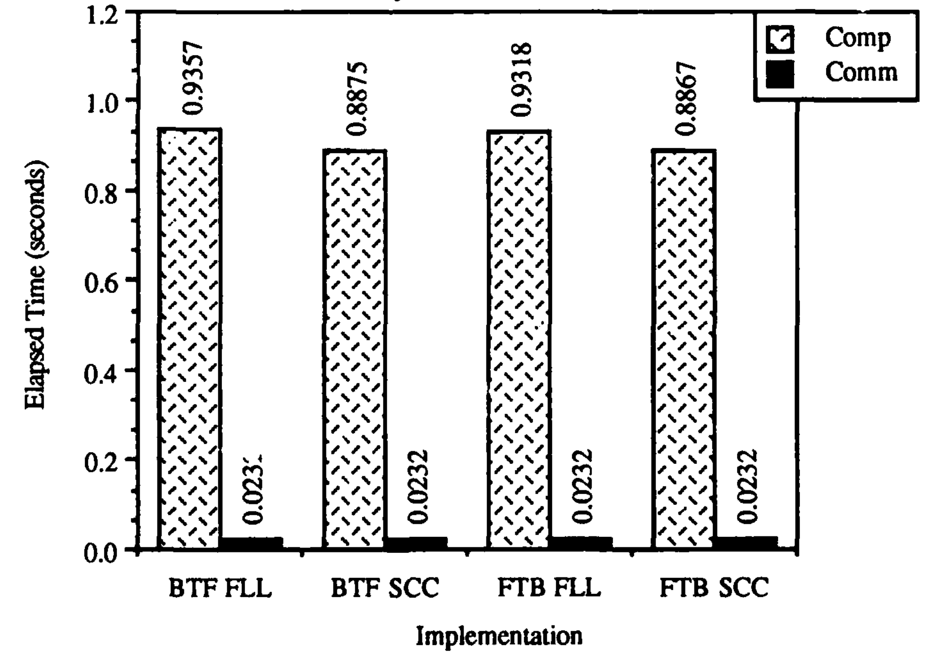


Abbreviated Pipeline Performance
Mesh Topology Results
16k Byte Packets
Elapsed Time (seconds)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Comp
Comm
0.8977
0.0232
0.8415
0.0232
0.9409
0.0232
0.8958
0.0232
BTF FLL
BTF SCC
FTB FLL
FTB SCC
Implementation

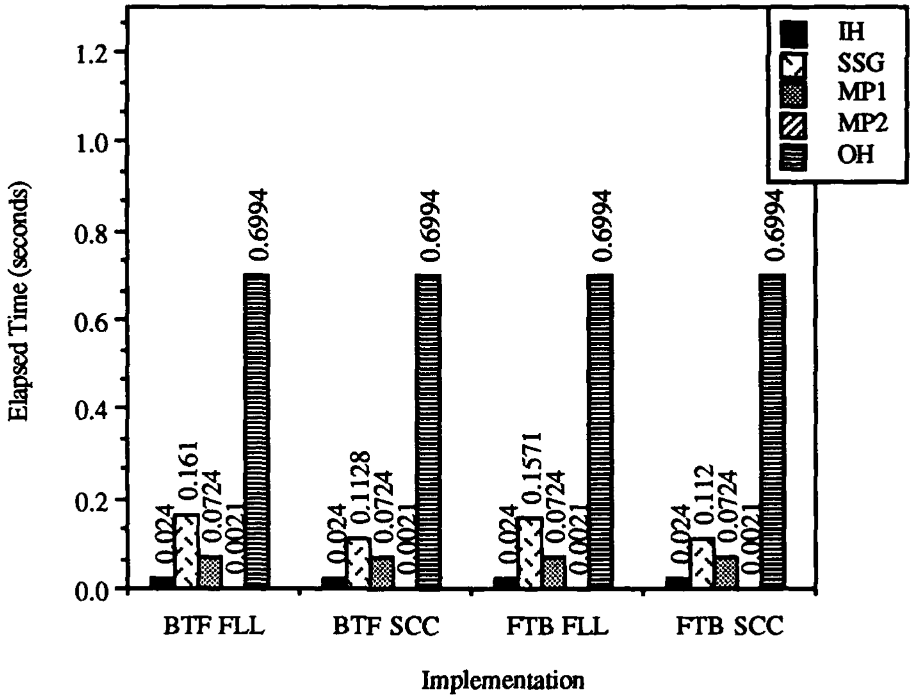
**Abbreviated Pipeline Performance
Mesh Topology Results
4k Byte Packets**



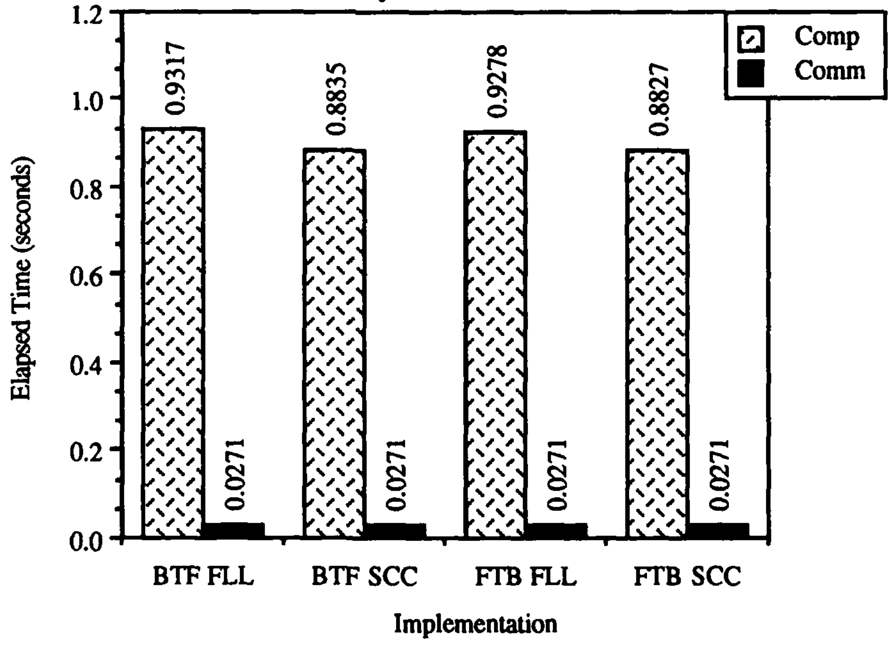
**Abbreviated Pipeline Performance
Mesh Topology Results
4k Byte Packets**



**Abbreviated Pipeline Performance
Mesh Topology Results
512 Byte Packets**



**Abbreviated Pipeline Performance
Mesh Topology Results
512 Byte Packets**



APPENDIX G

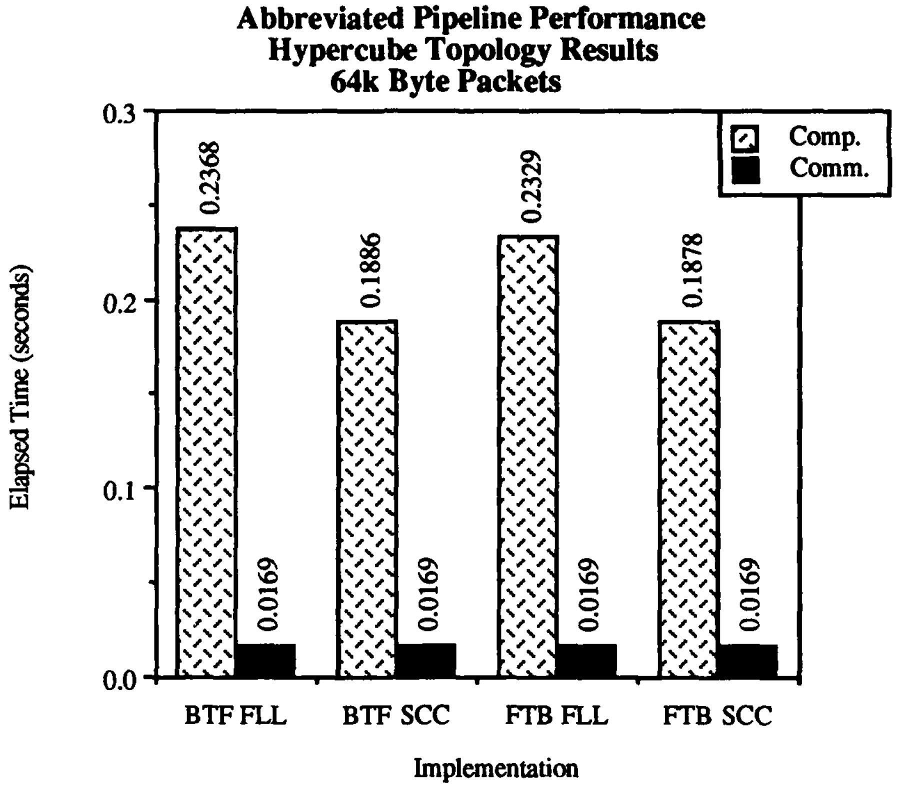
APPENDIX G

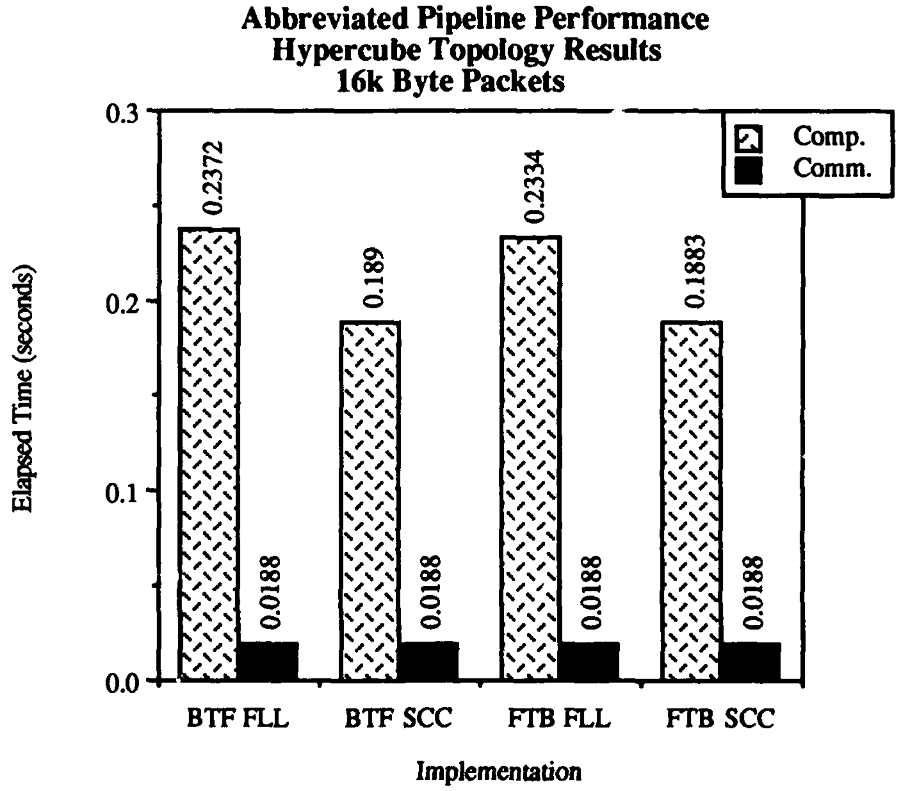
PIPELINE FRONT-END PERFORMANCE USING THE SCENE EDITING ALGORITHMS

G.1 INTRODUCTION.

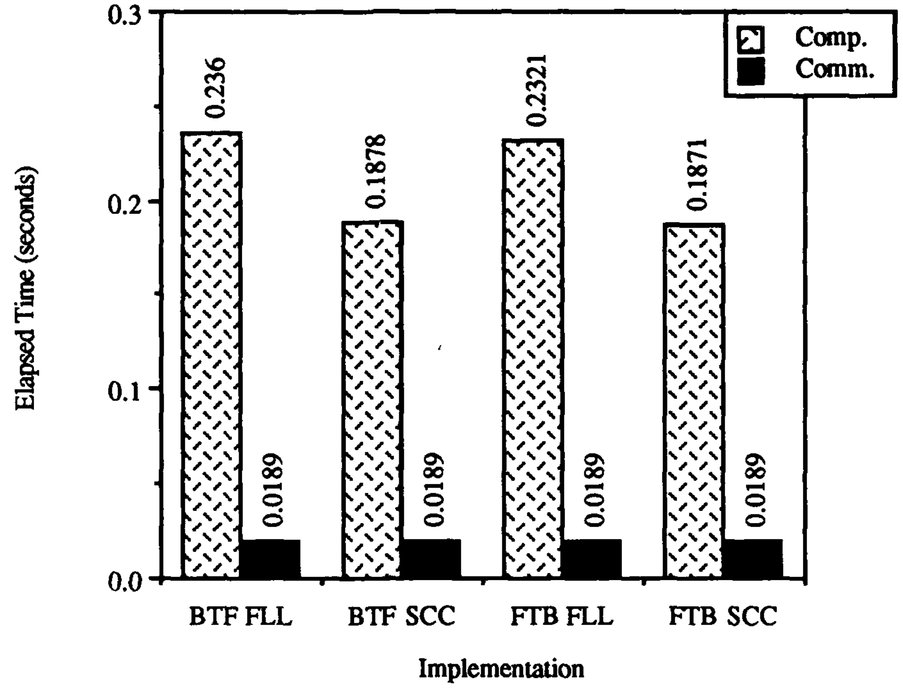
The performance results presented in this appendix depict the performance of the front-end stages of the abbreviated image processing pipeline using the scene editing recursive hidden-surface removal (HSR) algorithms described in Chapter V with the image shading time subtracted from the total computation time. This appendix highlights the throughput performance improvement obtained using the editing recursive HSR algorithm instead of the basic recursive HSR algorithm. Complete stage and throughput performance results for these implementations are contained in Appendix F. The performance of each algorithm is examined from two aspects. For each testbed machine's interconnection topology (mesh and hypercube) four packet sizes were tested, 512 byte, 4k byte, 16k byte, and 64k byte. For each packet size, the performance of the scene editing back-to-front (BTF) and front-to-back (FTB) hidden-surface removal (HSR) algorithms using either front-lower left coordinates (FLL) or subscene center coordinates (SCC) in the Sub-scene Generator (SSG) stage were tested. The performance results for each combination of interconnection topology and packet size are graphically presented for both the BTF and FTB HSR algorithms. Hypercube topology results for the 4 packet sizes in decreasing packet size order are provided first, followed by mesh topology results.

G.2 HYPERCUBE TOPOLOGY RESULTS.

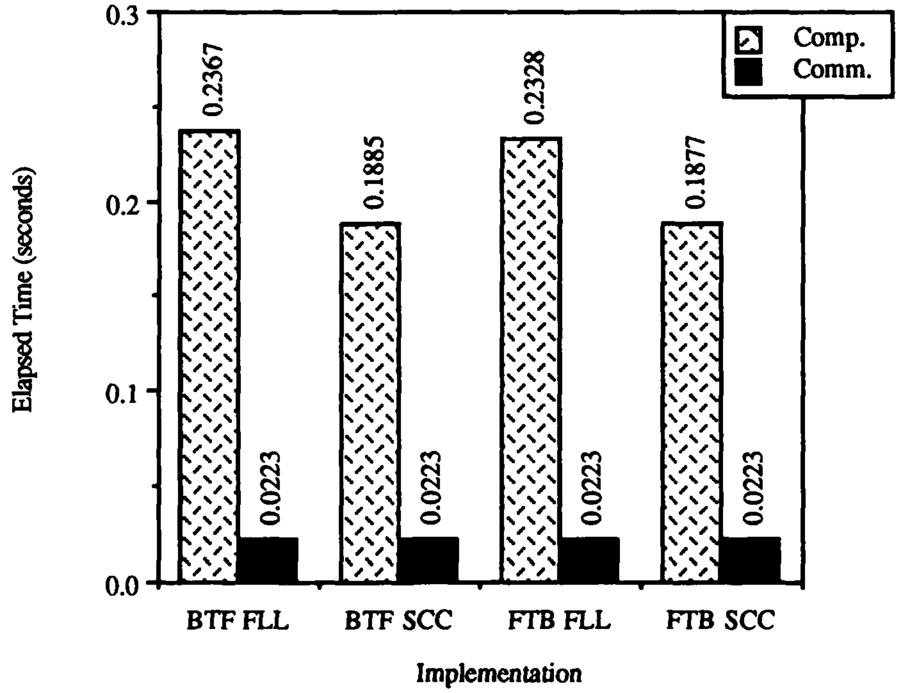




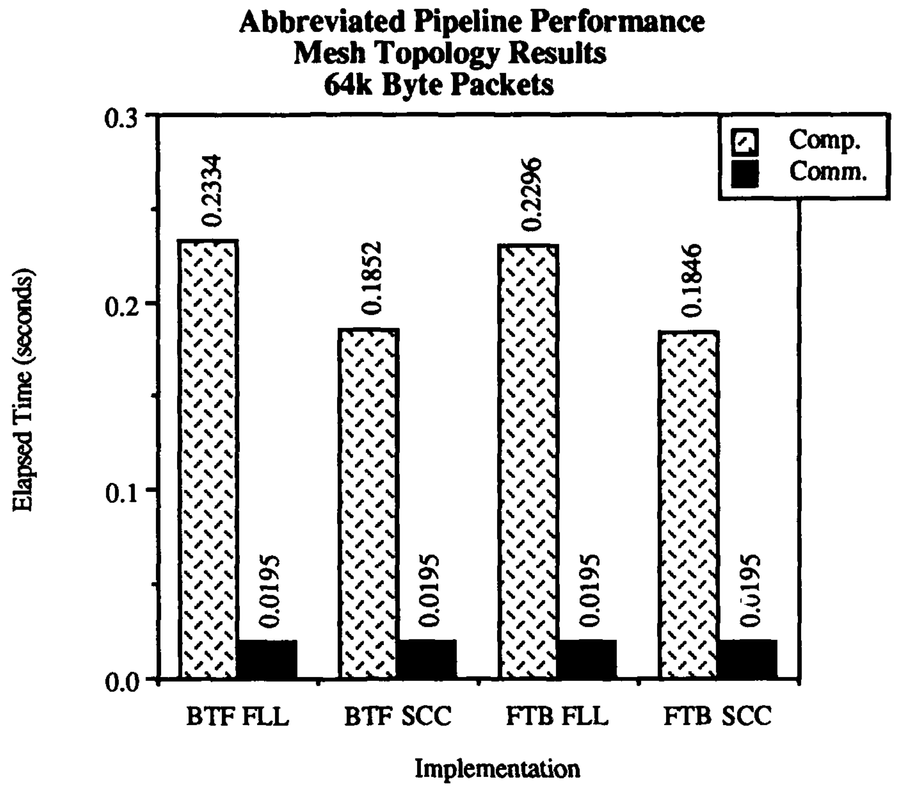
Abbreviated Pipeline Performance
Hypercube Topology Results
4k Byte Packets

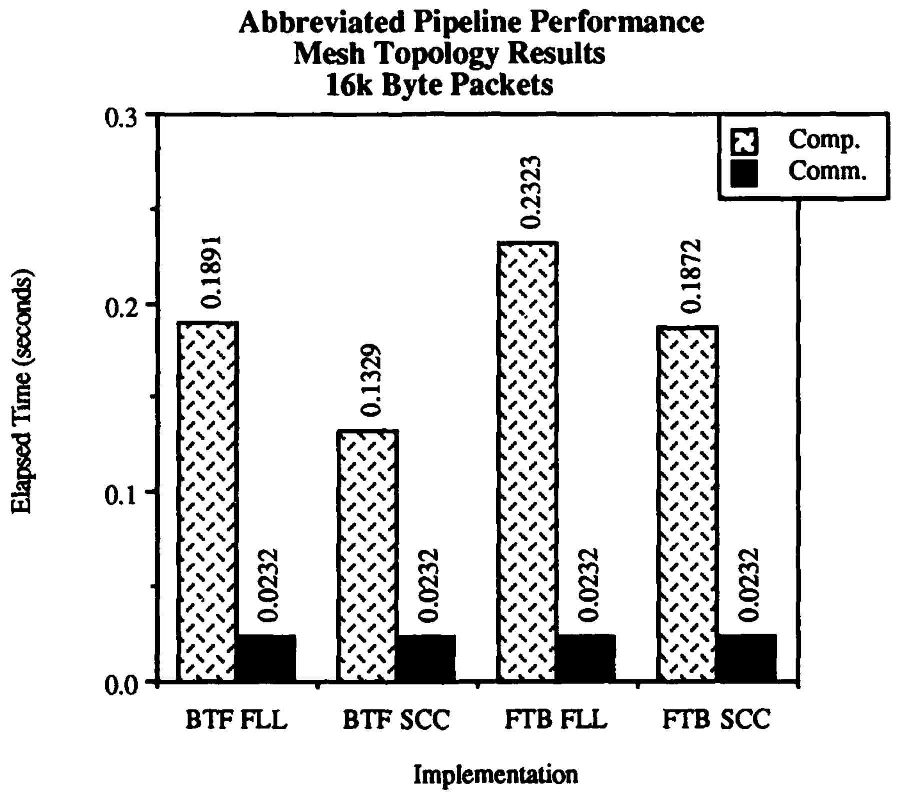


Abbreviated Pipeline Performance
Hypercube Topology Results
512 Byte Packets

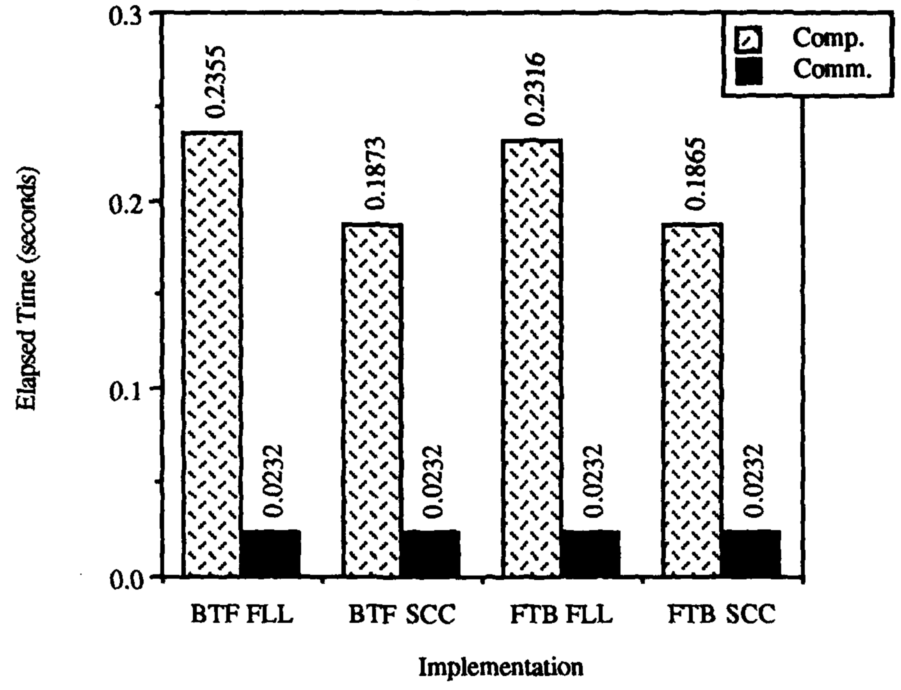


G.3 MESH TOPOLOGY RESULTS.

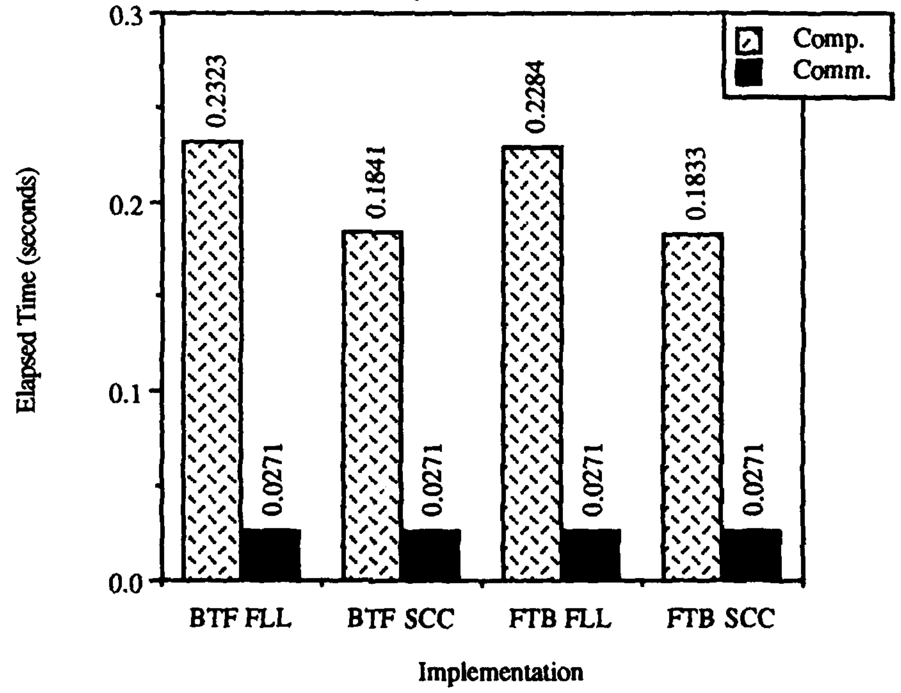




Abbreviated Pipeline Performance
Mesh Topology Results
4k Byte Packets



Abbreviated Pipeline Performance
Mesh Topology Results
512 Byte Packets



APPENDIX H

APPENDIX H
THE ADAPTIVE TERMINATION BACK-TO-FRONT RECURSIVE HIDDEN-SURFACE REMOVAL ALGORITHMS

H.1 INTRODUCTION.

This appendix presents the adaptive termination back-to-front recursive hidden-surface removal algorithms using the octant numbering scheme presented in Figure 3.2. Depending upon whether the implementation uses a sub-cube center or front-lower left criteria for selecting the $\{\mathcal{P}_O\}$ set, one of two different adjustments to the algorithm must be made at the deepest level of recursion in order to generate correct image space and object space coordinate values. Rather than complicate the presentation of the algorithm with these conditions, they are described here with the understanding that they are engaged when required. First, because for a front-lower left (FLL) $\{\mathcal{P}_O\}$ rotation is best performed by rotating the scene about the origin without translating the scene center to the origin, negative values for image space coordinates, and hence image array indices, are produced at various rotations. To insure that the image is written into the image output buffer, an image array offset equal to N is added to each image array coordinate. The image array is correspondingly expanded to a size of 2N x 2N. For a sub-cube center (SCC) $\{\mathcal{P}_O\}$, an image array index offset is not required. Second, for a SCC $\{\mathcal{P}_O\}$, the algorithm produces the set of coordinates at sub-cube octant 3 and not the sub-cube center at recursion level m-1. This mis-positioning can be corrected by adjusting each of the coordinates generated at level m-1 by applying R twice to $\{\mathcal{V}_{I_3}\}$ level m-1 and subtracting the resultant vector from

each coordinate set at level m-1. The FLL technique does not have this problem as it is designed to produce coordinate values from FLL octant coordinate values and not sub-cube center values.

H.2 THE ADAPTIVELY TERMINATING BACK-TO-FRONT ALGORITHM.

Given:

a normal resolution scene size of N x N x N where $N = 2^m$
volume center or front-lower left coordinates $\{V_c\} = \{x_c, y_c, z_c\}$
object space octant generation coordinate set $\{O\} = \{\{O_{0_x}, O_{0_y}, O_{0_z}\} \ldots \{O_{7_x}, O_{7_y}, O_{7_z}\}$
map - a function which returns the original object space octant number given an image space octant number
object_array - size N x N x N which contains a 3D image
image_array - size 2N x 2N
Plane_Coeffs - the coefficients of the equation which positions the cutting plane in image space
behind(I_x, I_y, I_z, Plane_Coeffs) - a function which determines if the given image space coordinates are lying on or behind the cutting plane.
BTFRP - the image space plane equation coefficients for the Back-to-Front Reference Plane, positioned parallel to the cutting plane and at a distance -5N along the z-axis from it.
Plane_Distance(I_x, I_y, I_z, Plane_Coeffs) - a function which returns the distance from the cutting plane to image space coordinates I_x, I_y, I_z
BTFRP_Distance(I_x, I_y, I_z, BTFRP) - a function which returns the distance between the BTFRP and image space coordinates I_x, I_y, I_z
BTFRP_Plane_Distance(Plane_Coeffs,BTFRP) - a function which returns the distance between the BTFRP and the cutting plane
L - length of the x-axis projection of the vector running from an octant's generation point to the furthest point in the octant, equal to N/2 for the FLL approach and N/4 for the SCC approach
object_rootptr - a pointer to the root node of an oct-tree, points to an array of eight other pointers, each entry of which contains a pointer to an oct-tree node containing an array of eight pointers
object_leaf - the oct-tree leaf structure which contains a voxel density value for the corresponding volume in object space & entry in the object_array
Window - a user defined upper and lower threshold for the voxel values which may appear in the rendered image

```
recurse_build_objecttree($x_c$, $y_c$, $z_c$, object_rootptr,1)
until FOREVER do
  {
  acquire_user_display_requirements(x_rot, y_rot, z_rot, Plane_Coeffs, Window)
/* compute image space volume center coordinates {$V_i$} */
  compute_image_space_coordinates($x_c$,$y_c$,$z_c$, $x'_c$, $y'_c$, $z'_c$, x_rot, y_rot, z_rot)
/* determine image space location of BTFRP, at distance -5N from cutting plane */
  position_BTFRP(Plane_Coeffs, BTFRP)
/* compute image space octant center coordinate set {I} */
  for s = 0 to 7 do
  compute_image_space_coordinates($O_{s_x}$, $O_{s_y}$, $O_{s_z}$, $I_{s_x}$, $I_{s_y}$, $I_{s_z}$,x_rot, y_rot, z_rot)
/* sort image space coordinates sets into back to front order, with deepest
   octant's coordinates in leading set position */
  back-to-front_sort({I})
/* recursively compute all image space coordinates */
  recurse_display(I, object_rootptr, $x'_c$, $y'_c$, $z'_c$, 1)
  }

recurse_display(I, objecttree_ptr, $C_{i_x}$, $C_{i_y}$ , $C_{i_z}$, recursion_level)
  {
  for j = 0 to 7 do
  {
  k = map(j)
/* compute new image space octant center coordinates */
  if j = (1,3,5,7)   then $C_{x'} = C_{i_x} + (I_{j_x}$ >> recursion_level[1])
                     else $C_{x'} = C_{i_x} - (I_{j_x}$ >> recursion_level)
  if j = (2,3,6,7)   then $C_{y'} = C_{i_y} + (I_{j_y}$ >> recursion_level)
                     else $C_{y'} = C_{i_y} - (I_{j_y}$ >> recursion_level)
  if j = (0,1,2,3)   then $C_{z'} = C_{i_z} + (I_{j_z}$ >> recursion_level)
                     else $C_{z'} = C_{i_z} - (I_{j_z}$ >> recursion_level)


/* check to determine if current image space coordinate is further from BTFRP than
           the cutting plane or if cutting plane passes through the current image space
           octant */

  if recursion_level < $\log_2(N)$ and
           (BTFRP_Plane_Distance(Plane_Coeffs,BTFRP)<
                          BTFRP_Distance($C_{x'}$,$C_{y'}$,$C_{z'}$,BTFRP)) or
           (Plane_Distance($C_{x'}$,$C_{y'}$,$C_{z'}$,Plane_Coeffs) <
                          $\sqrt{(L>>(recursion\_level-1))^2 * 3)}$
  then
     recurse_display(I, objecttree_ptr->branch_array[k],
                          $C_{x'}$, $C_{y'}$ , $C_{z'}$, recursion_level+1)
```

[1]>> is defined to be the right-shift operator.

```
 if (recursion_level = $\log_2(N)$) and
         (objecttree_ptr->branch_array[k]->object_leaf $\supset$ Window)
 then
/* are at the deepest branch of the object tree, so to achieve supersampling just plot
 the eight image space points using the one required object space leaf's value */
    for h = 0 to 7 do
    {
    /* compute new image space octant center coordinates */
    if h = (1,3,5,7)        then $C_{x''} = C_{x'} + (I_{h_x} >> m+1)$
         else $C_{x''} = C_{x'} - (I_{h_x} >> m+1)$
    if h = (2,3,6,7)        then $C_{y''} = C_{y'} + (I_{h_y} >> m+1)$
         else $C_{y''} = C_{y'} - (I_{h_y} >> m+1)$
    if h = (0,1,2,3)        then $C_{z''} = C_{z'} + (I_{h_z} >> m+1)$
         else $C_{z''} = C_{z'} - (I_{h_z} >> m+1)$

    if *behind*($C_{x'}$, $C_{y'}$ , $C_{z'}$, Plane_Coeffs) then
         image_array[$C_{x''}$][$C_{y''}$] = objecttree_ptr->branch_array[k]->object_leaf
      }
 }}


*recurse_build_objecttree*($C_{o_x}$, $C_{o_y}$, $C_{o_z}$, objecttree_ptr,recursion_level)
{
/* compute the object space coordinates (which are identical to the object_array
 values) for each of the octants */
 for k = 0 to 7 do
 {
 if k = (1,3,5,7)  then $C_x = C_{o_x} + (O_{k_x} >>$ recursion_level)
          else $C_x = C_{o_x} - (O_{k_x} >>$ recursion_level)
 if k = (2,3,6,7)  then $C_y = C_{o_y} + (O_{k_y} >>$ recursion_level)
          else $C_y = C_{o_y} - (O_{k_y} >>$ recursion_level)
 if k = (0,1,2,3)  then $C_z = C_{o_z} + (O_{k_z} >>$ recursion_level)
             else $C_z = C_{o_z} - (O_{k_z} >>$ recursion_level)

/* if are at a level in the oct-tree where branch nodes are required, create a new
 branch node in the object space oct-tree holding an array of eight pointers */
 if recursion_level < $\log_2(N)$ then
                 objecttree_ptr->branch_array[k]->new_objecttree_node
/* if are at a level in the oct-tree where leaf nodes are required, create a new leaf
 node in the object space oct-tree holding an integer density value */
 if recursion_level = $\log_2(N)$ then
                 objecttree_ptr->branch_array[k]->object_leaf
```

```
if recursion_level < $\log_2(N)$
        then     /* pass coordinates and pointer to next level in oct-tree */
          recurse_build_objecttree($C_x$, $C_y$, $C_z$, objecttree_ptr->branch_array[k],
                                   recursion_level+1)

if recursion_level = $\log_2(N)$   /* record the object array value in the oct-tree leaf */
        then
        objecttree_ptr->branch_array[k]-> object_leaf  = object_array [$C_x$][$C_y$][$C_z$]

}}
```

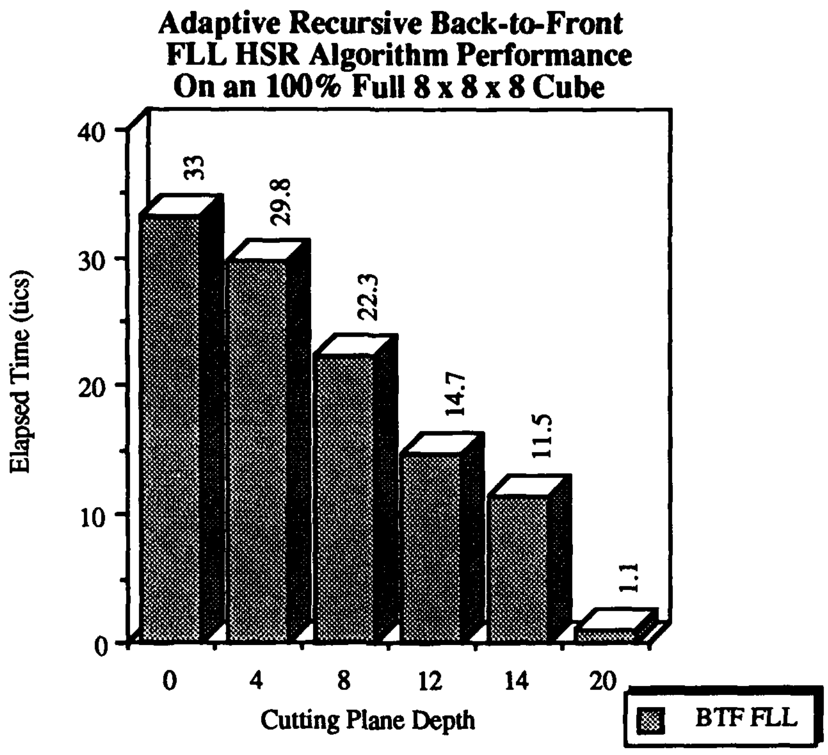
APPENDIX I

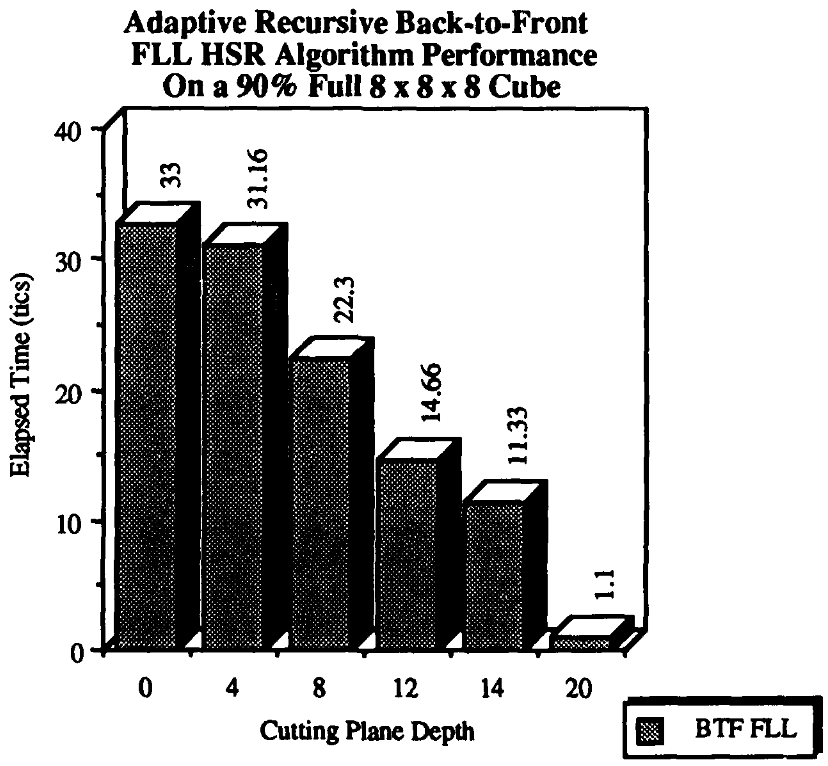
APPENDIX I
PERFORMANCE OF THE ADAPTIVE RECURSIVE BACK-TO-FRONT HIDDEN-SURFACE REMOVAL ALGORITHMS

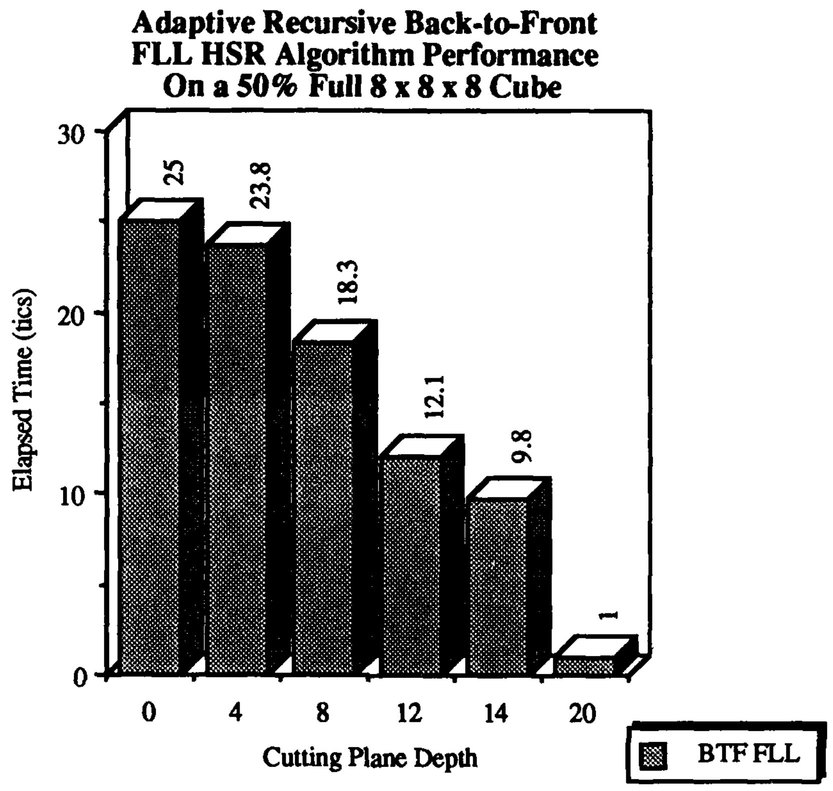
I.1 INTRODUCTION.

The performance results presented in this appendix depict the performance of the adaptive recursive back-to-front (BTF) hidden-surface removal (HSR) algorithm on a range of scene sizes and threshold window saturation values. The graphs in this appendix also compare the performance of the adaptive BTF HSR algorithm with the editing recursive HSR algorithm and the basic recursive HSR algorithm. For each cube (scene) size, the performance of the adaptive back-to-front (BTF) hidden-surface removal algorithm using either front-lower left coordinates (FLL) or subscene center coordinates (SCC) as the $\{\mathcal{P}_0\}$ were tested. The presentation is organized with the algorithm's performance on 8^3, 16^3, and 32^3 scene sizes across a range of threshold window saturations shown first to highlight the change in execution speed of each variant of the algorithm as cutting plane penetration varies, with the results for each combination of scene size, saturation, and algorithm variant presented in a series of graphs. This is followed by a series of graphs which depict the relative performance of the adaptive, basic, and editing BTF HSR algorithms across the same range of scene sizes. In both sequences of graphs, the FLL variant's results are presented first followed by the SCC variant

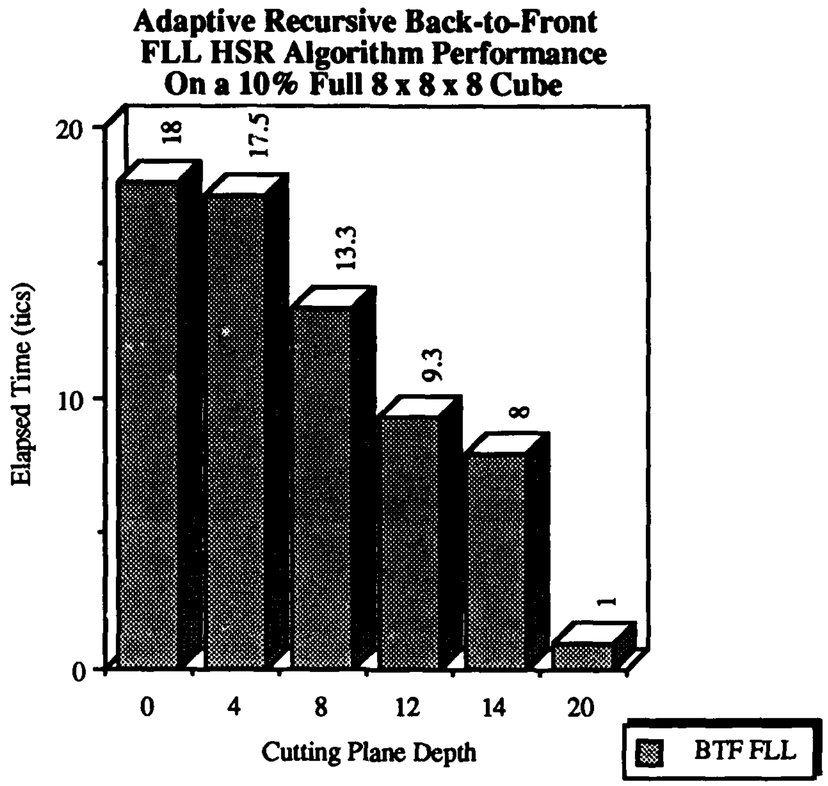
I.2 ADAPTIVE RECURSIVE BTF FLL HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN AN 8 X 8 X 8 CUBE.



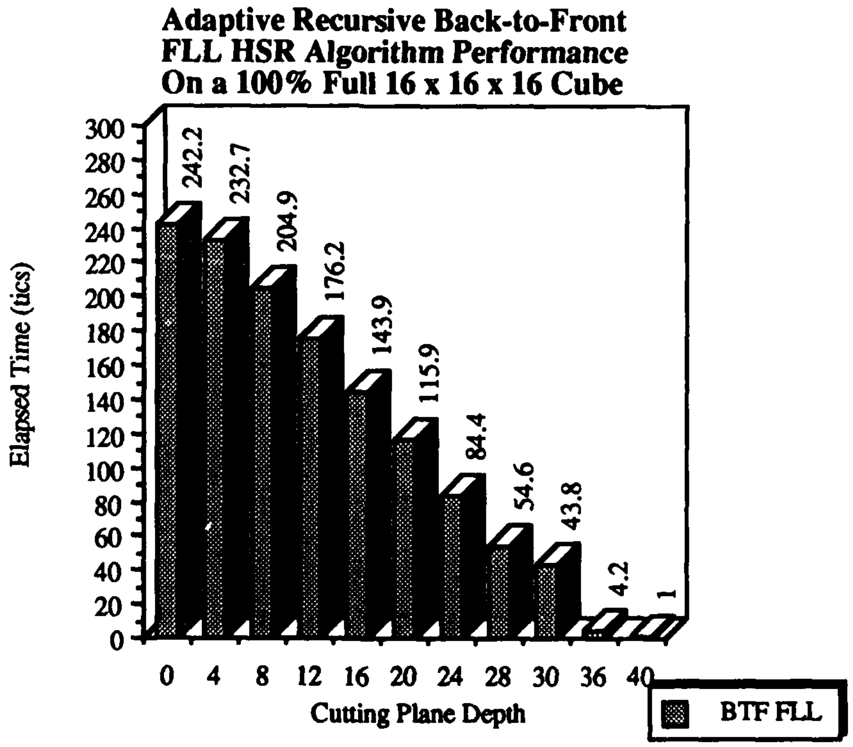




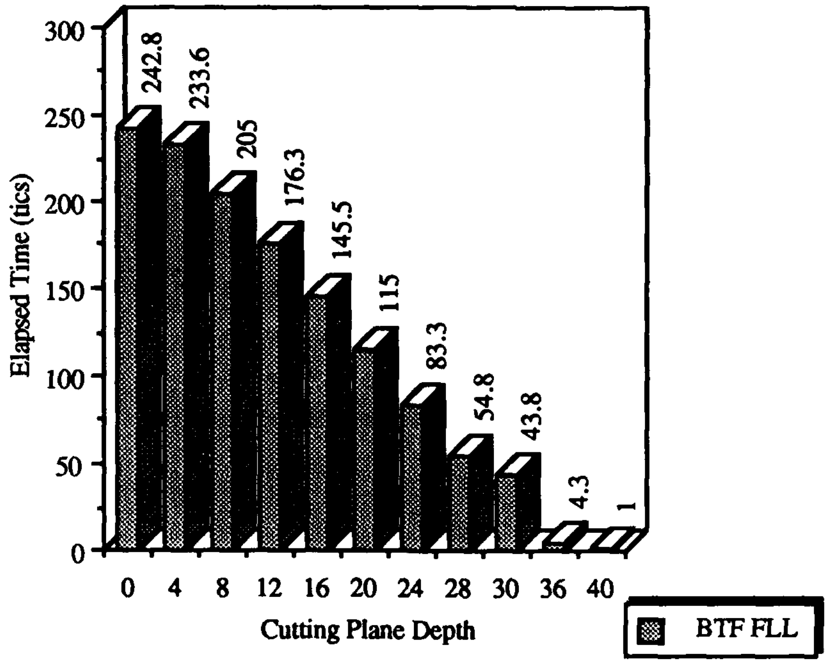
Adaptive Recursive Back-to-Front
FLL HSR Algorithm Performance
On a 50% Full 8 x 8 x 8 Cube
Elapsed Time (tics)
30
20
10
0
25
23.8
18.3
12.1
9.8
1
0
4
8
12
14
20
Cutting Plane Depth
BTF FLL



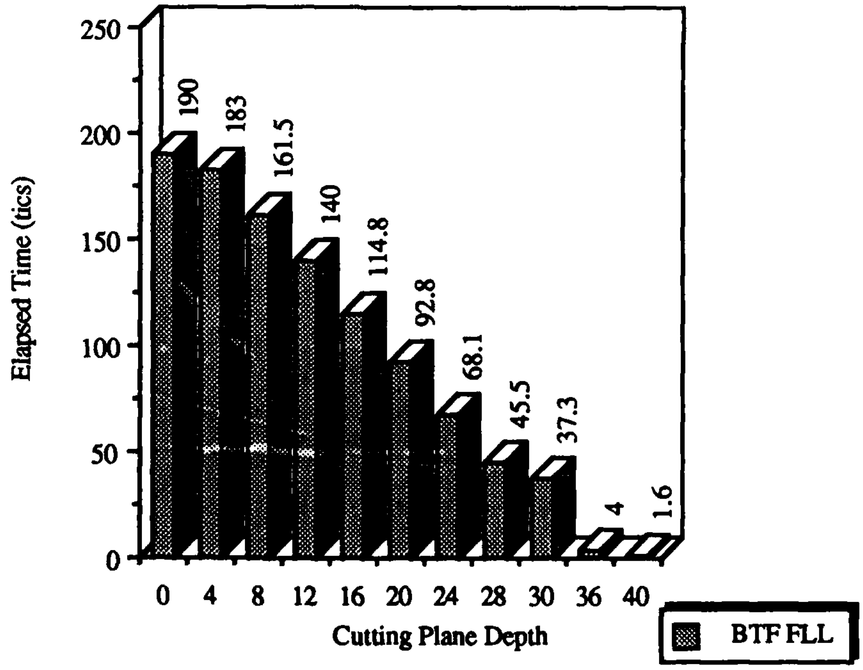
I.3 ADAPTIVE RECURSIVE BTF FLL HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN A 16 X 16 X 16 CUBE.

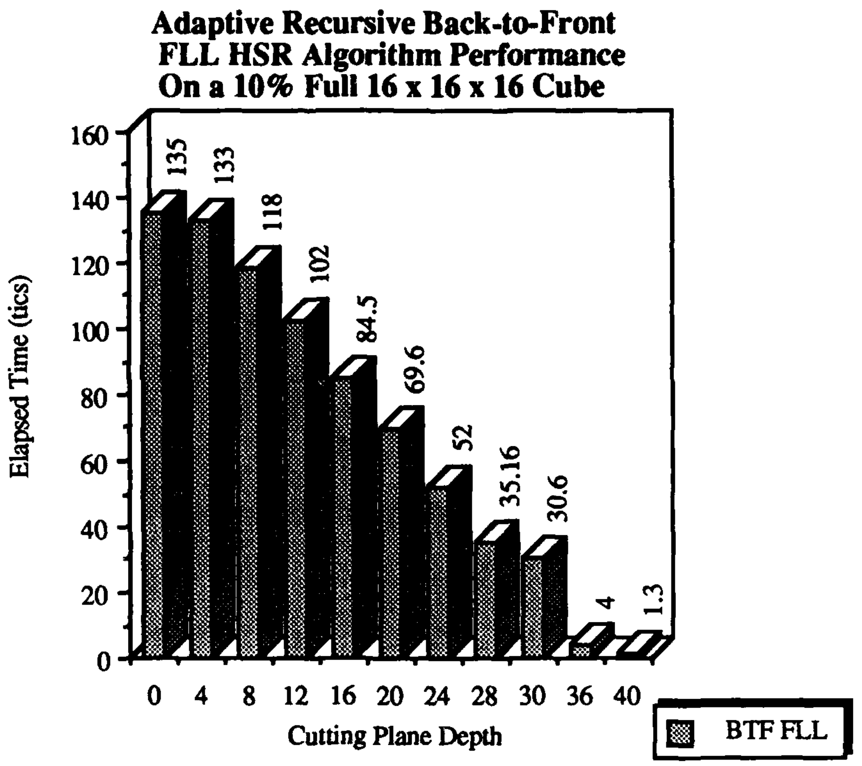


Adaptive Recursive Back-to-Front FLL HSR Algorithm Performance On a 90% Full 16 x 16 x 16 Cube

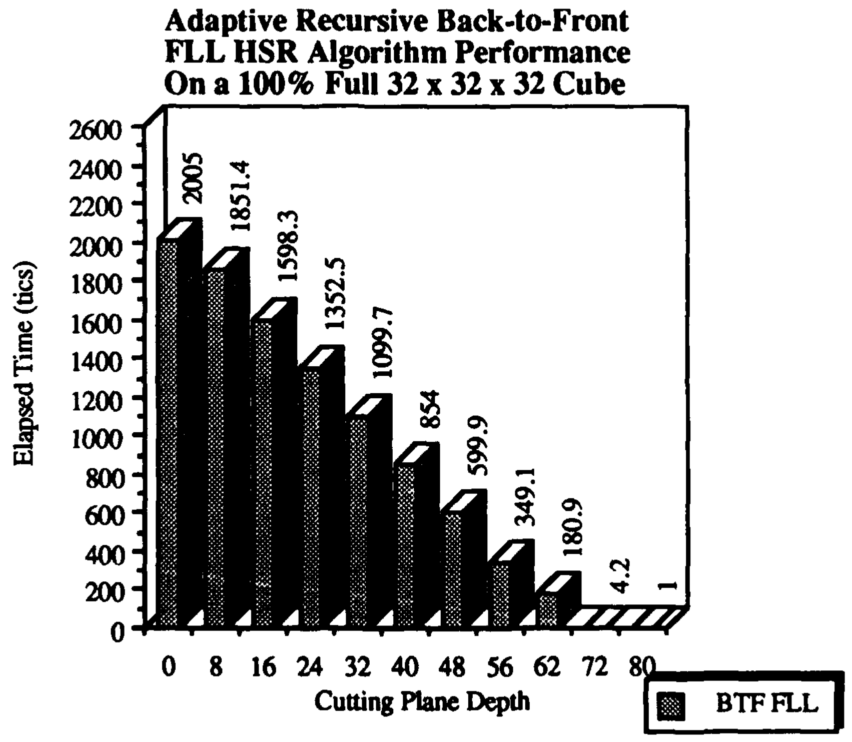


Adaptive Recursive Back-to-Front FLL HSR Algorithm Performance On a 50% Full 16 x 16 x 16 Cube

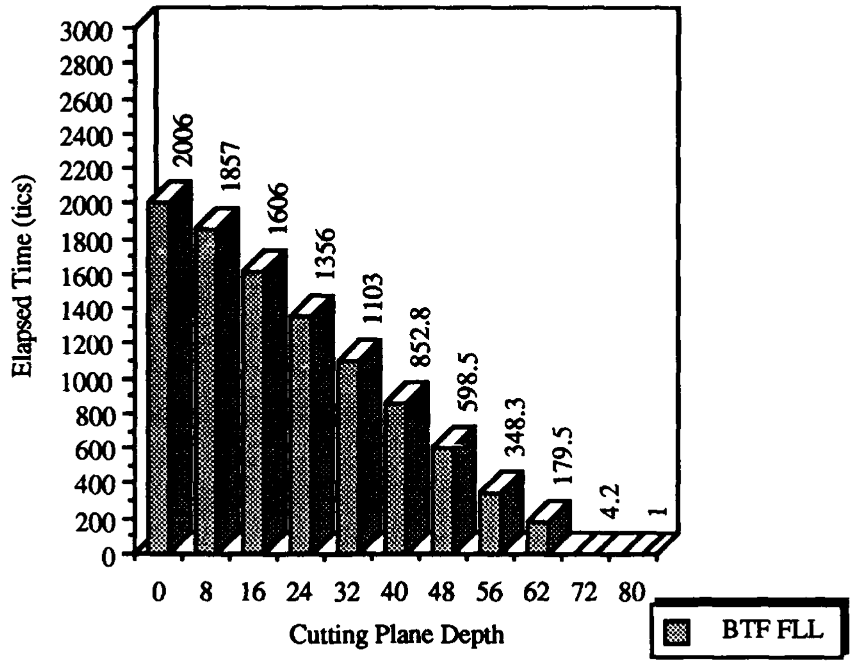




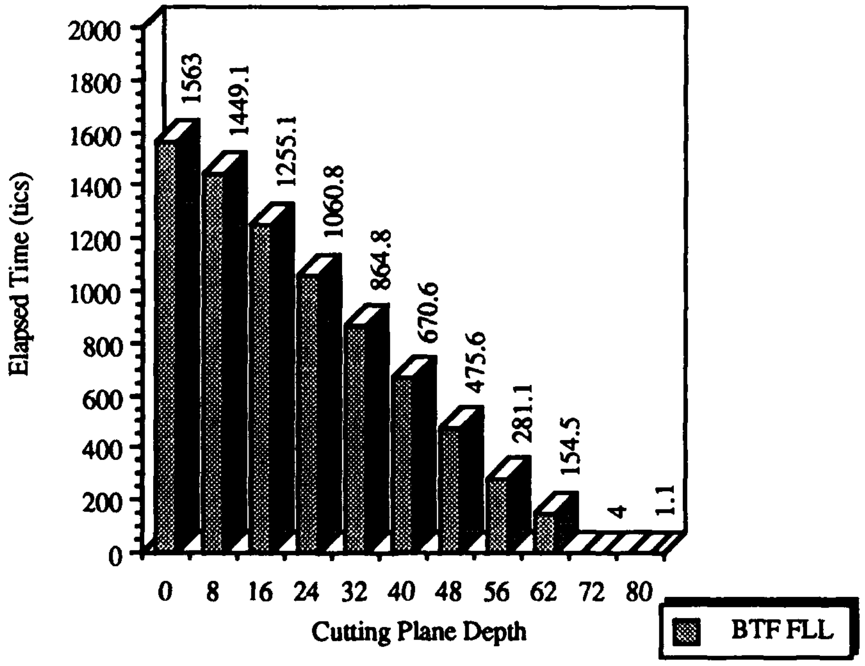
I.4 ADAPTIVE RECURSIVE BTF FLL HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN A 32 X 32 X 32 CUBE.



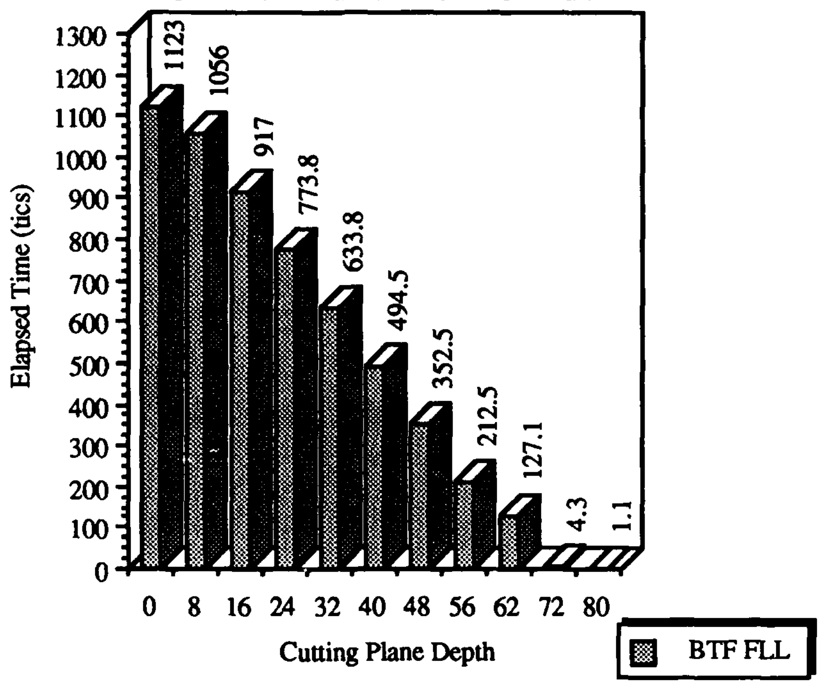
Adaptive Recursive Back-to-Front FLL HSR Algorithm Performance On a 90% Full 32 x 32 x 32 Cube



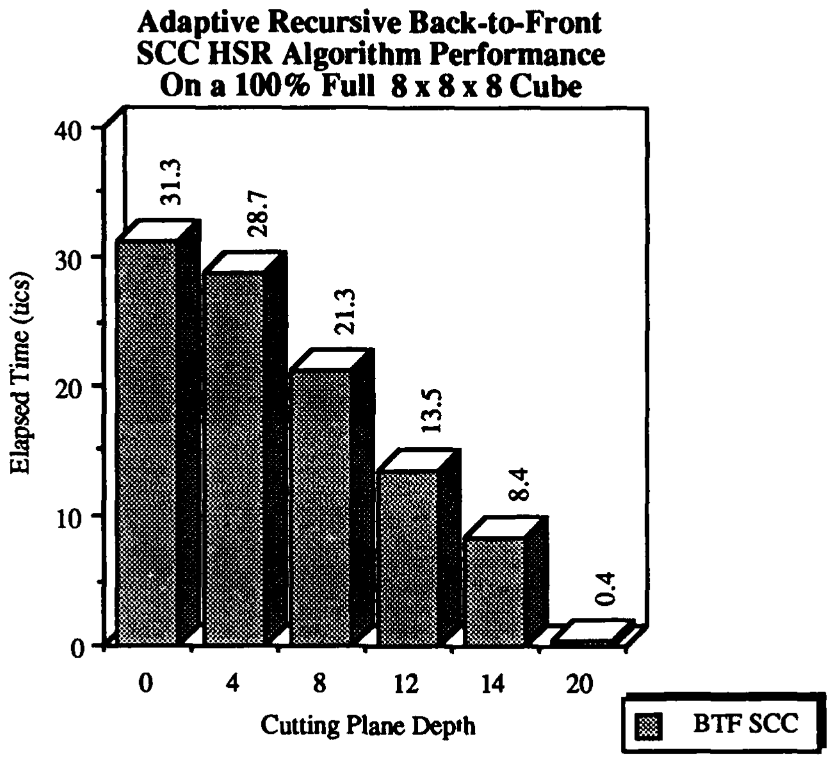
Adaptive Recursive Back-to-Front FLL HSR Algorithm Performance On a 50% Full 32 x 32 x 32 Cube

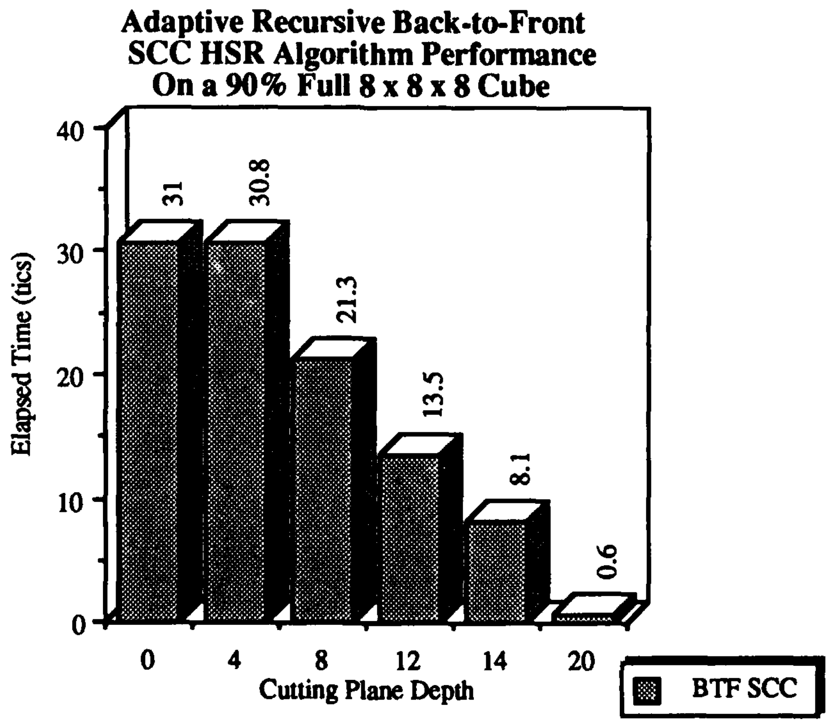


Adaptive Recursive Back-to-Front FLL HSR Algorithm Performance On a 10% Full 32 x 32 x 32 Cube

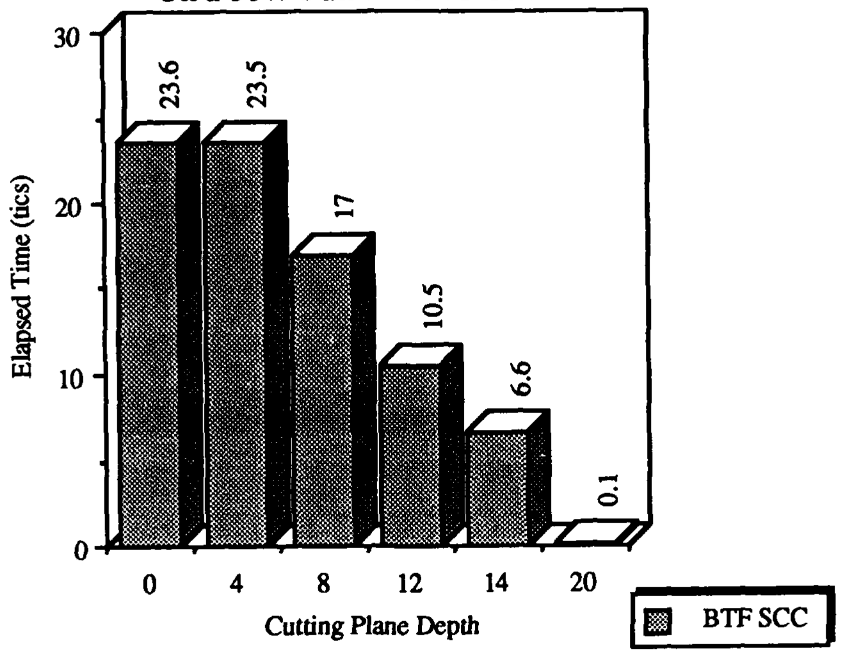


I.5 ADAPTIVE RECURSIVE BTF SCC HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN AN 8 X 8 X 8 CUBE.

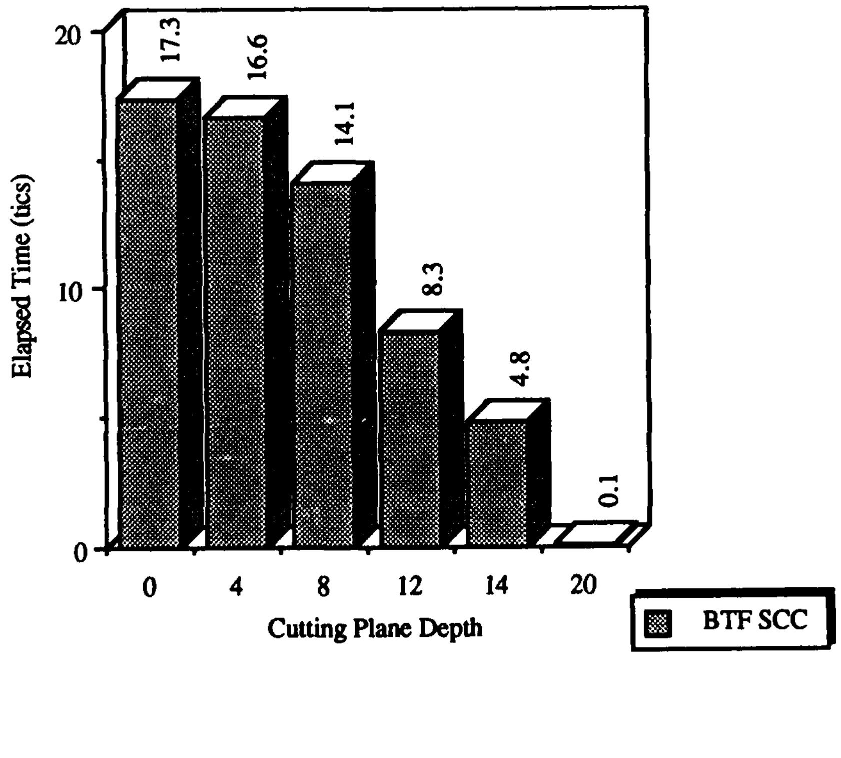




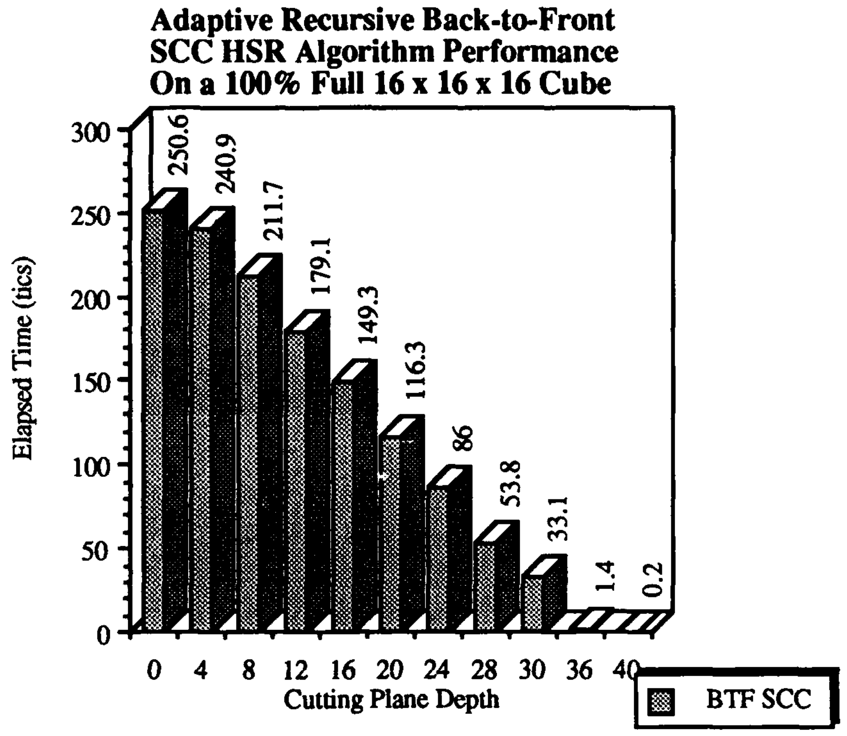
Adaptive Recursive Back-to-Front SCC HSR Algorithm Performance On a 50% Full 8 x 8 x 8 Cube

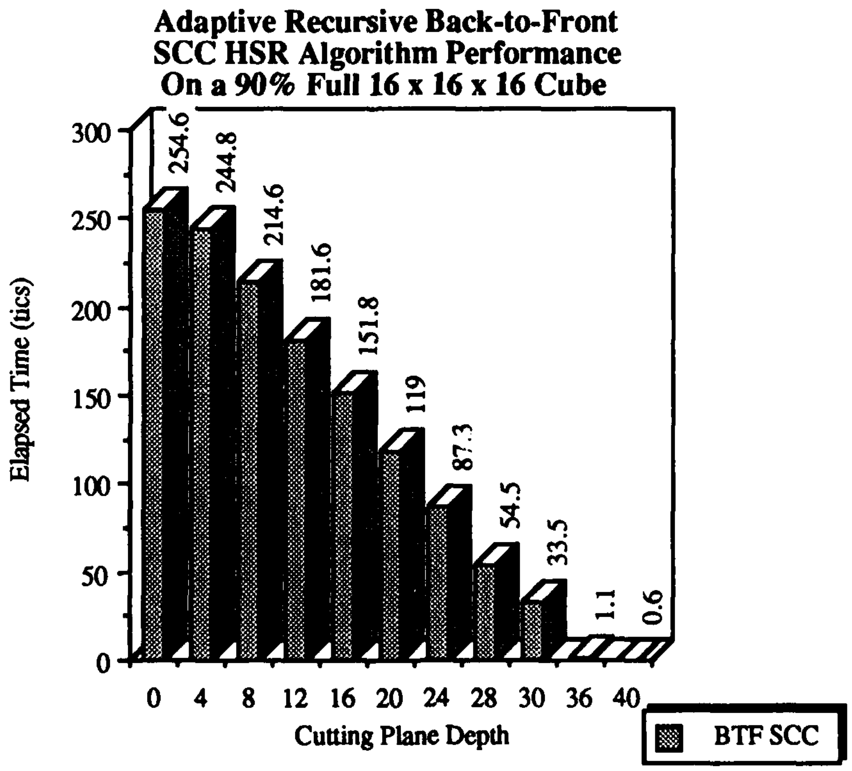


Adaptive Recursive Back-to-Front SCC HSR Algorithm Performance On a 10% Full 8 x 8 x 8 Cube

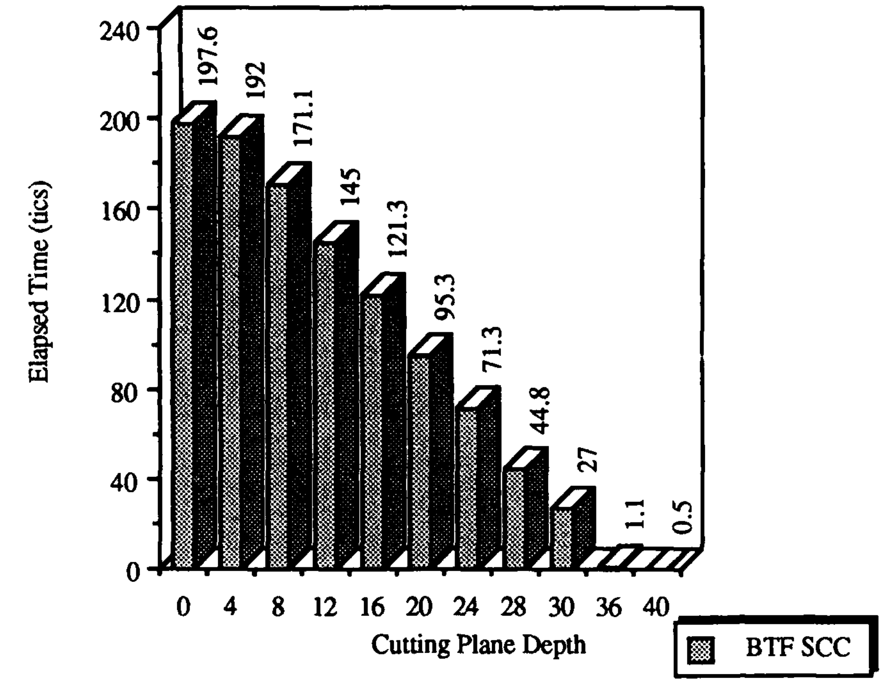


I.6 ADAPTIVE RECURSIVE BTF SCC HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN A 16 X 16 X 16 CUBE.

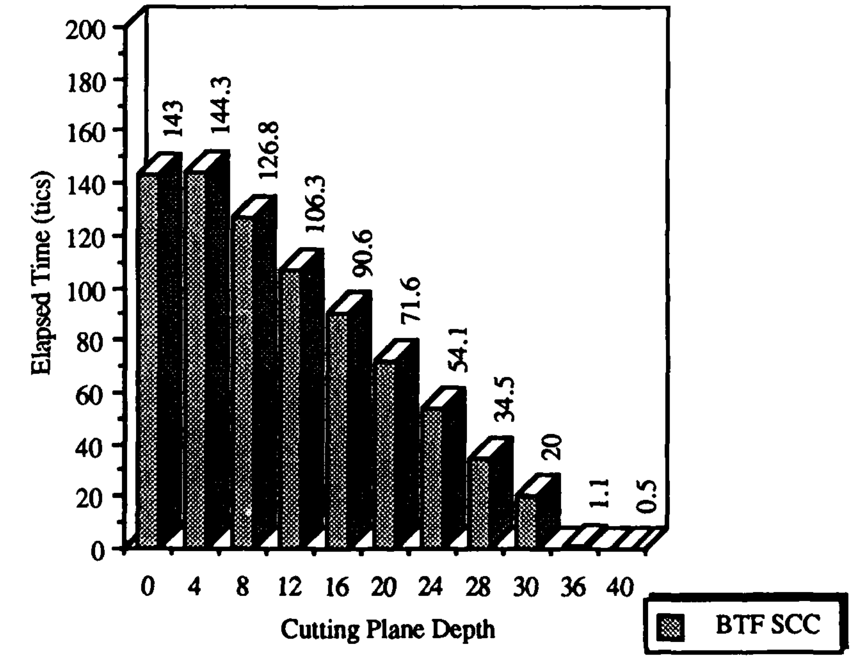




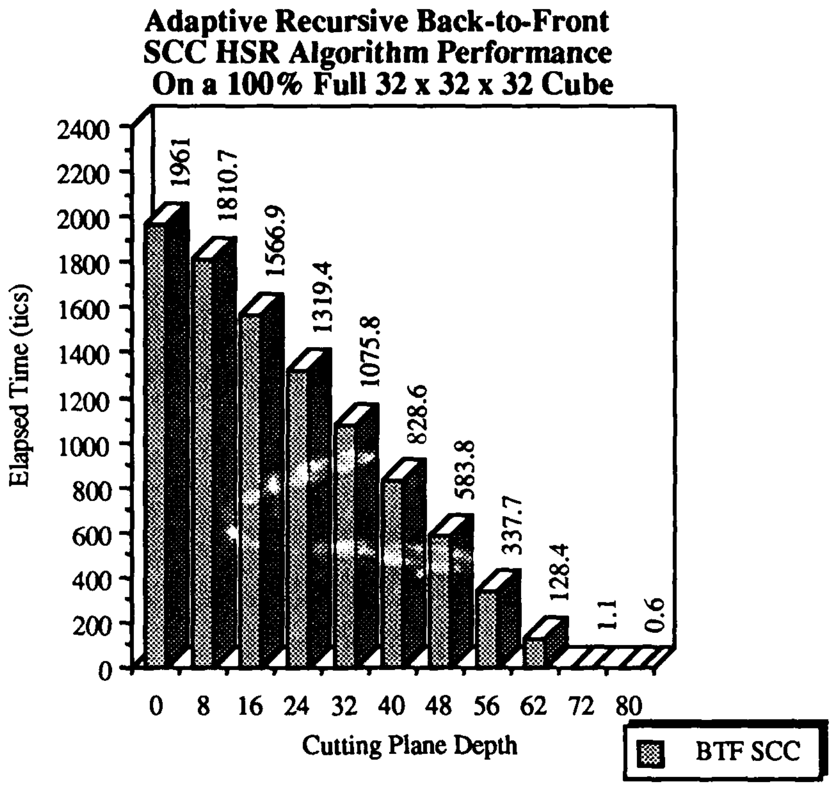
Adaptive Recursivc Back-to-Front SCC HSR Algorithm Performance On a 50% Full 16 x 16 x 16 Cube

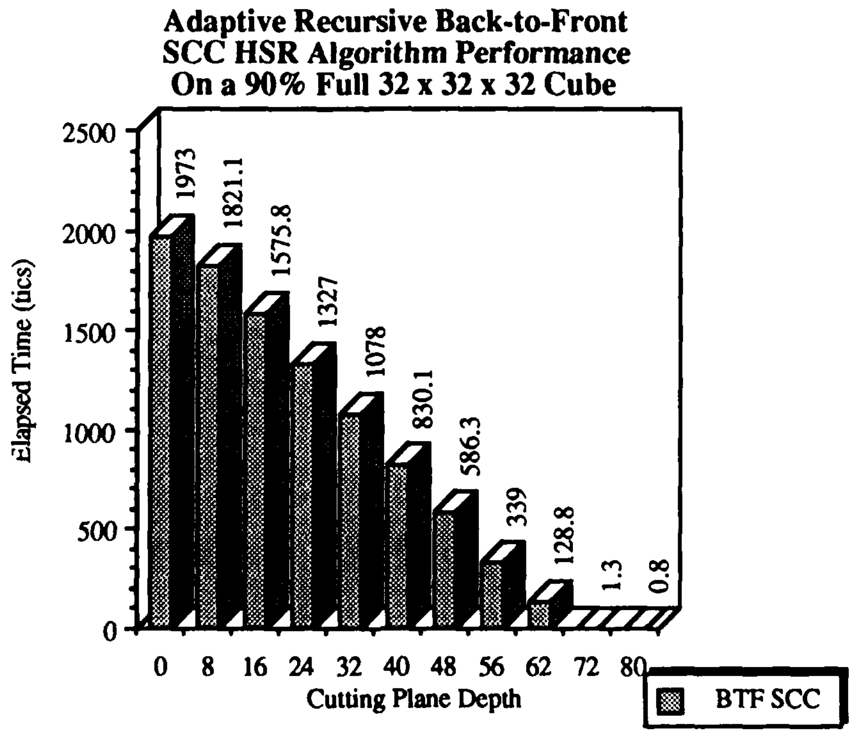


Adaptive Recursive Back-to-Front SCC HSR Algorithm Performance On a 10% Full 16 x 16 x 16 Cube

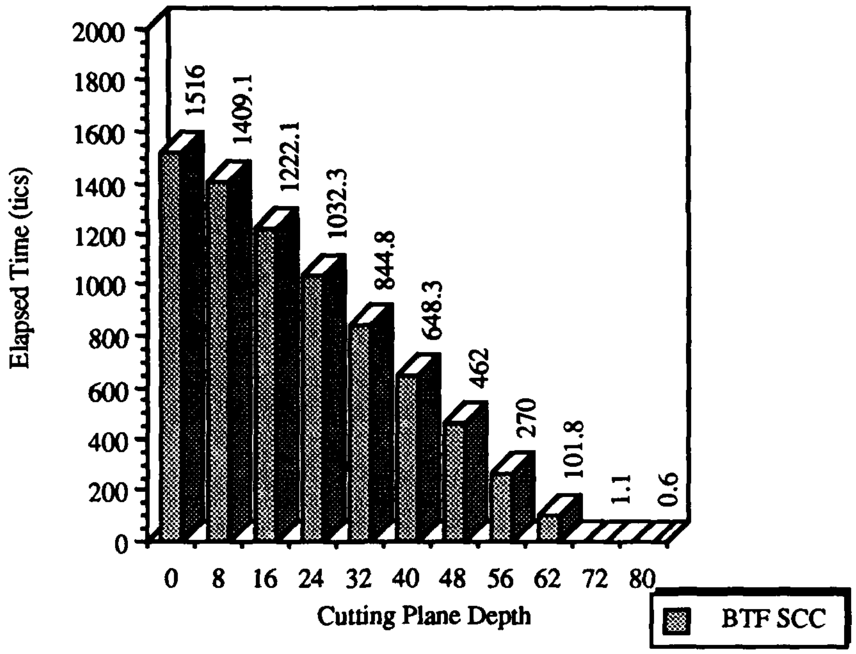


I.7 ADAPTIVE RECURSIVE BTF SCC HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN A 32 X 32 X 32 CUBE.

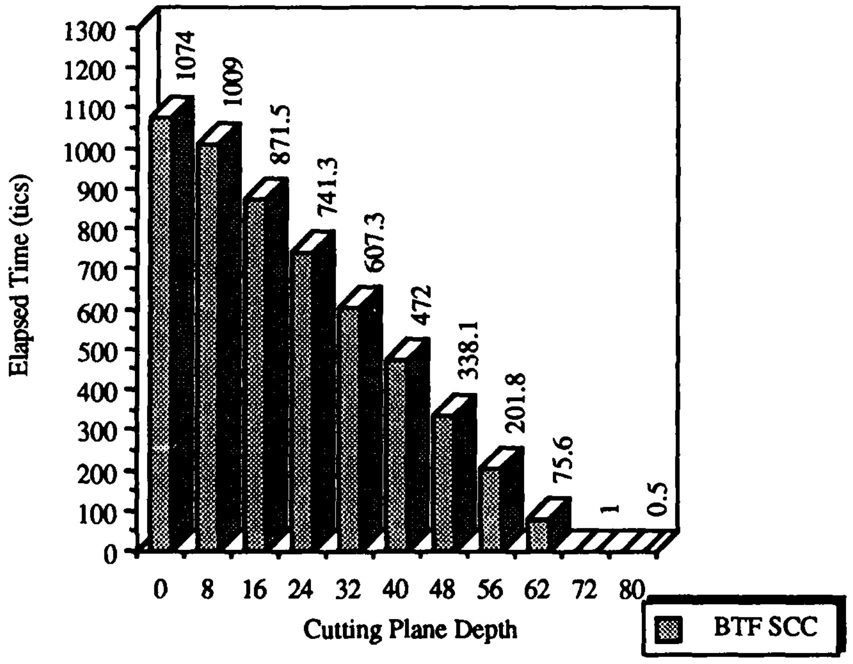




Adaptive Recursive Back-to-Front
SCC HSR Algorithm Performance
On a 50% Full 32 x 32 x 32 Cube

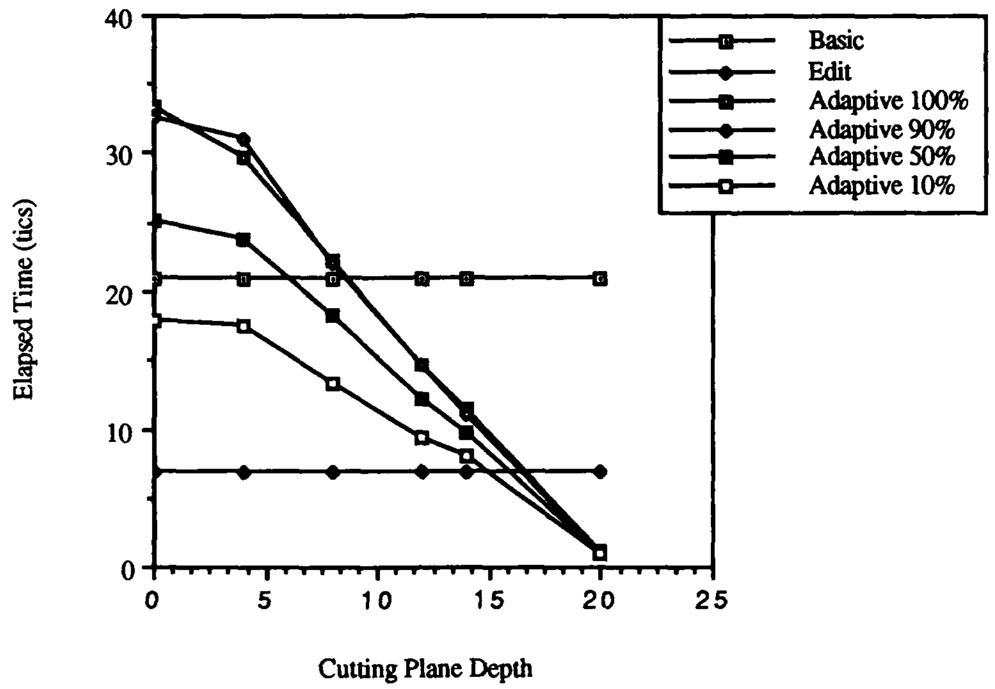


Adaptive Recursive Back-to-Front
SCC HSR Algorithm Performance
On a 10% Full 32 x 32 x 32 Cube

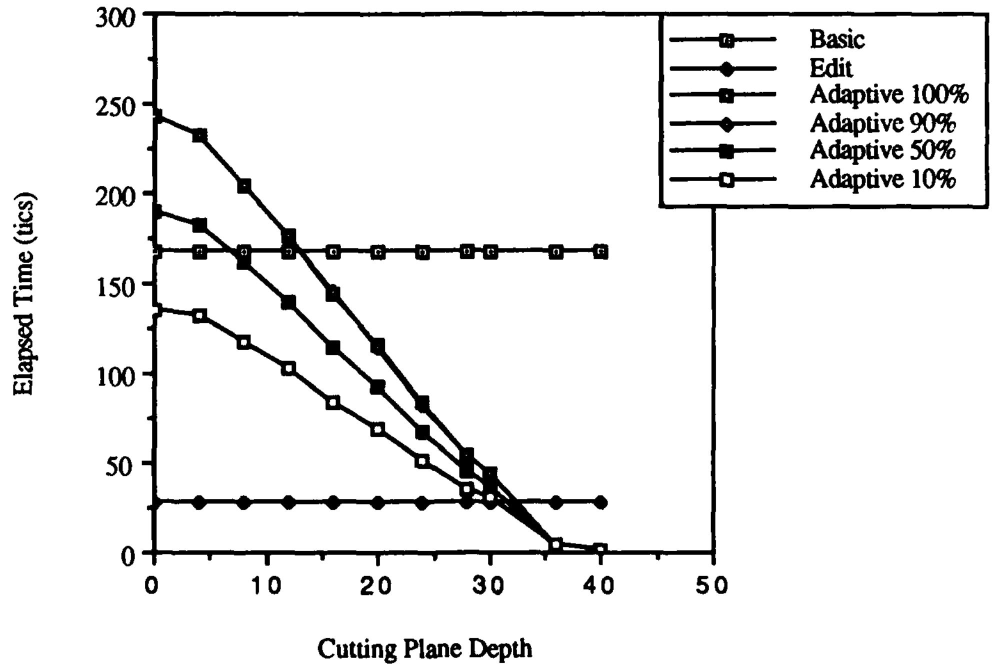


I.8 ADAPTIVE RECURSIVE BTF HSR ALGORITHM PERFORMANCE COMPARED TO EDITING AND BASIC BTF HSR ALGORITHMS.

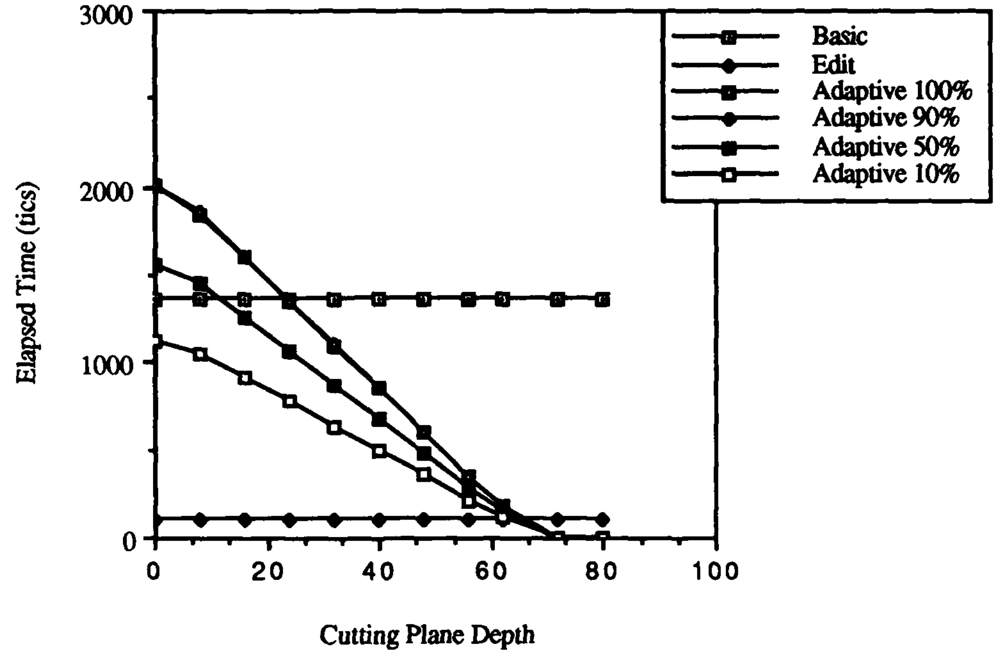
Comparison of Basic, Edit, and Adaptive BTF FLL HSR Algorithms On an 8 x 8 x 8 Cube at Different Thresholds



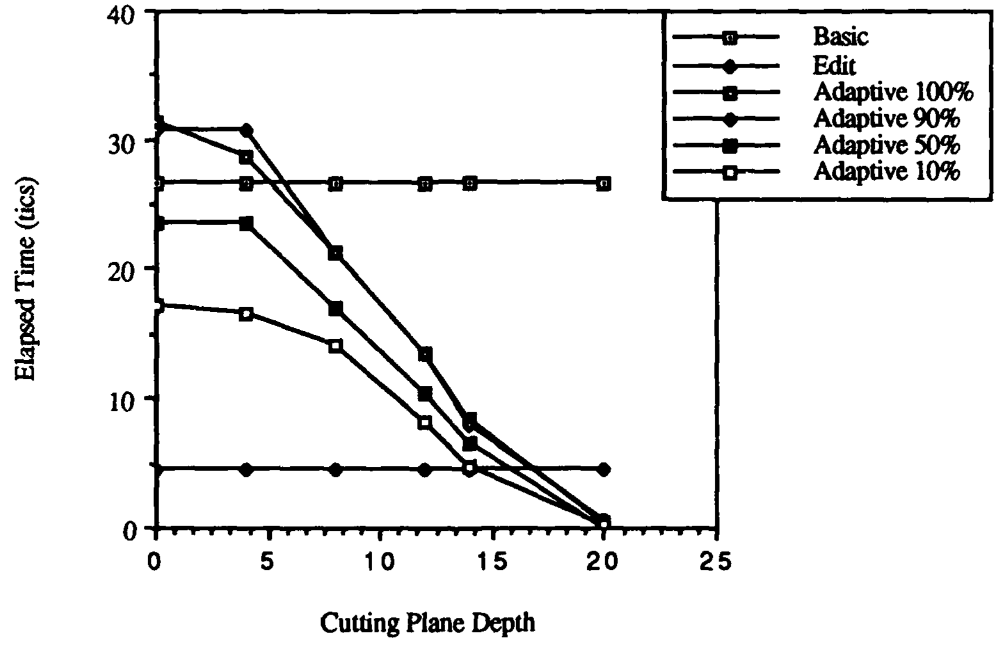
Comparison of Basic, Edit, and Adaptive BTF FLL HSR Algorithms On a 16 x 16 x 16 Cube at Different Thresholds



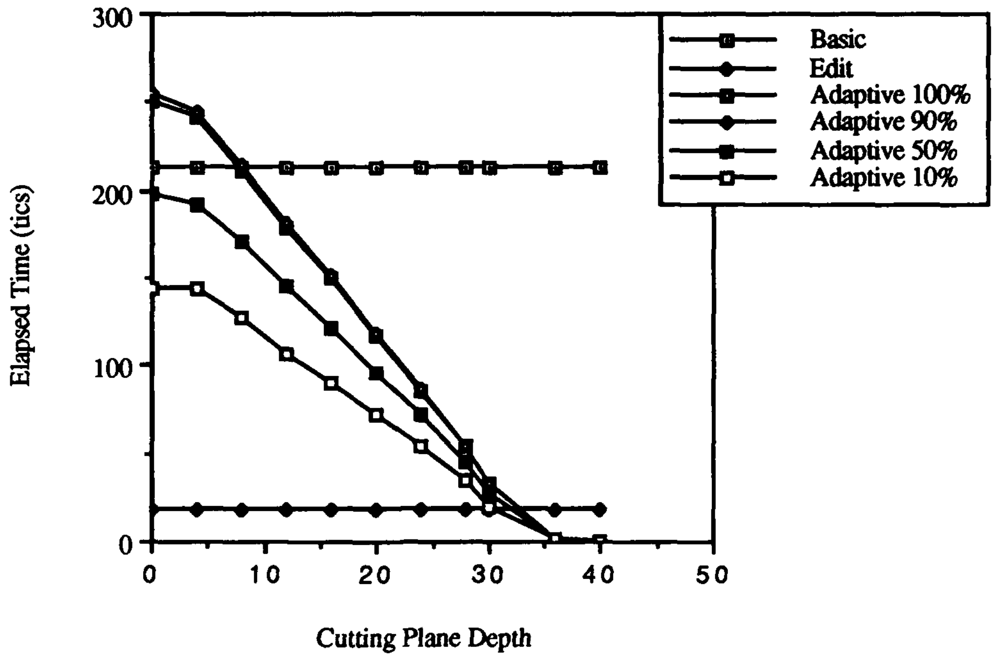
Comparison of Basic, Edit, and Adaptive BTF FLL HSR Algorithms On a 32 x 32 x 32 Cube at Different Thresholds



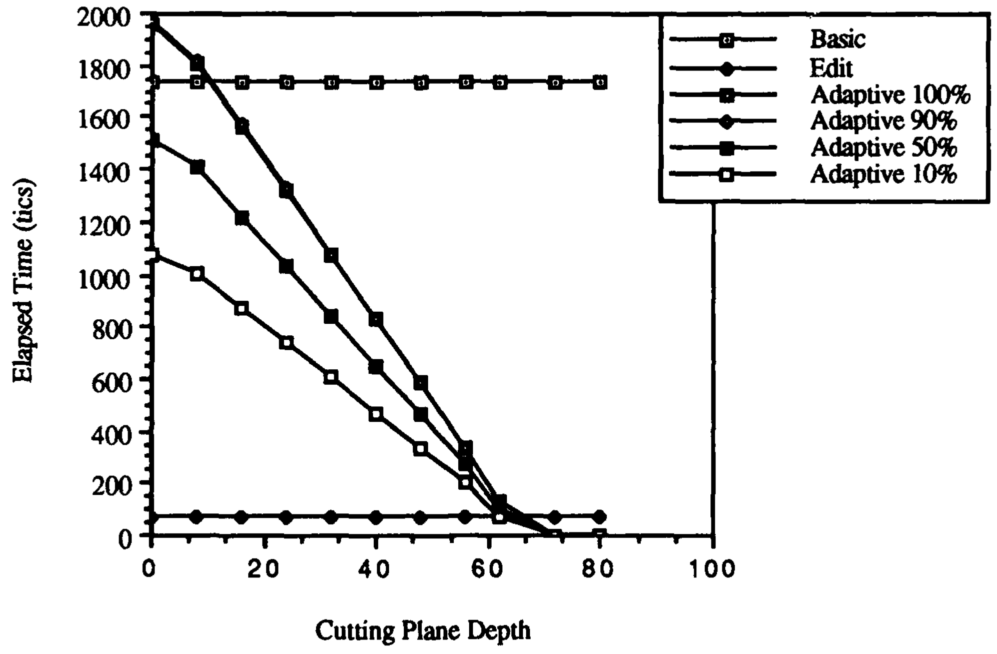
Comparison of Basic, Edit, and Adaptive BTF SCC HSR Algorithms On an 8 x 8 x 8 Cube at Different Thresholds



Comparison of Basic, Edit, and Adaptive BTF SCC HSR Algorithms On a 16 x 16 x 16 Cube at Different Thresholds



Comparison of Basic, Edit, and Adaptive BTF SCC HSR Algorithms On a 32 x 32 x 32 Cube at Different Thresholds



APPENDIX J

APPENDIX J
THE ADAPTIVE TERMINATION FRONT-TO-BACK RECURSIVE HIDDEN-SURFACE REMOVAL ALGORITHMS

J.1 INTRODUCTION.

This appendix presents the adaptive termination front-to-back recursive hidden-surface removal algorithms using the octant numbering scheme presented in Figure 3.2. Depending upon whether the implementation uses a sub-cube center or front-lower left criteria for selecting the $\{\mathcal{P}_O\}$ set, one of two different adjustments to the algorithm must be made at the deepest level of recursion in order to generate correct image space and object space coordinate values. Rather than complicate the presentation of the algorithm with these conditions, they are described here with the understanding that they are engaged when required. First, because for a front-lower left (FLL) $\{\mathcal{P}_O\}$ rotation is best performed by rotating the scene about the origin without translating the scene center to the origin, negative values for image space coordinates, and hence image array indices, are produced at various rotations. To insure that the image is written into the image output buffer, an image array offset equal to N is added to each image array coordinate. The image array is correspondingly expanded to a size of 2N x 2N. For a sub-cube center (SCC) $\{\mathcal{P}_O\}$, an image array index offset is not required. Second, for a SCC $\{\mathcal{P}_O\}$, the algorithm produces the set of coordinates at sub-cube octant 3 and not the sub-cube center at recursion level m-1. This mis-positioning can be corrected by adjusting each of the coordinates generated at level m-1 by applying R twice to $\{\mathcal{V}_{I_3}\}$ level m-1 and subtracting the resultant vector from

each coordinate set at level m-1. The FLL technique does not have this problem as it is designed to produce coordinate values from FLL octant coordinate values and not sub-cube center values.

J.2 THE ADAPTIVELY TERMINATING FRONT-TO-BACK ALGORITHM.

Given:

a normal resolution scene size of N x N x N where $N = 2^m$
volume center or front-lower left coordinates $\{V_c\} = \{x_c, y_c, z_c\}$
object space octant generation coordinate set $\{O\} = \{\{O_{0_x}, O_{0_y}, O_{0_z}\} \ldots \{O_{7_x}, O_{7_y}, O_{7_z}\}$
map - a function which returns the original object space octant number given an image space octant number
object_array - size N x N x N which contains a 3D image
image_array - size 2N x 2N, initialized to -1
Plane_Coeffs - the coefficients of the equation which positions the cutting plane in image space
behind(I_x, I_y, I_z, Plane_Coeffs) - a function which determines if the given image space coordinates are lying on or behind the cutting plane.
object_rootptr - a pointer to the root node of an oct-tree, points to an array of eight other pointers, each entry of which contains a pointer to an oct-tree node containing an array of eight pointers
object_leaf - the oct-tree leaf structure which contains a voxel density value for the corresponding volume in object space & entry in the object_array
image_rootptr - the pointer to the root entry of image_tree.
image_tree - an oct-tree of depth $\log_2(N)+1$. Each branch node contains a count of the total number of positions in image_array filled in by the node (node_count), the array examined (see below) which indicates if the corresponding image space has ever been checked, and pointers to each of the eight child nodes stored in an array called child_ptrarray. The leaves of the image space tree contain the total count for each of the eight image space voxels they represent and the image space coordinates of the generation point for the terminal octant
image_leaf - a structure used in image_tree to contain the total number of image space voxels in a terminal image space octant written to screen space and the image space coordinates of the generation point for the terminal octant
examined - an array which indicates the leaf octants checked so that only leafs octants skipped on the first pass are examined later, if necessary, not used for branch octants

project_into_image_tree - a function which determines the contribution of each image space octant and sub-octant to the final screen image. For this computation, octants are assumed to be solid, so no holes occur. Octants or sub-octants which are totally obscured are given a count of zero, otherwise a running total of the contribution of each octant/sub-octant to the final image is kept

form_image_tree - a function which builds the oct-tree that image space is projected into. Need only be built once

update_box - an array of size 2N x 2N which is used to determine the number of array elements each image space octant will project onto the screen, initialized to -1

projection_total - the number of voxel values to be written to screen space

Window - a user defined upper and lower threshold for the voxel values which may appear in the rendered image

```
/* build the oct-tree used for image space projection */
form_image_tree(image_rootptr,1)
recurse_build_objecttree($x_c$, $y_c$, $z_c$, object_rootptr,1)

until FOREVER do
  {
  acquire_user_display_requirements(x_rot, y_rot, z_rot, Plane_Coeffs, Window)
/* compute image space volume center coordinates {$V_i$} */
  compute_image_space_coordinates($x_c$,$y_c$,$z_c$, $x'_c$, $y'_c$, $z'_c$, x_rot, y_rot, z_rot)
/* compute image space octant center coordinate set {I} */
  for s = 0 to 7 do
  compute_image_space_coordinates($O_{s_x}$, $O_{s_y}$, $O_{s_z}$, $I_{s_x}$, $I_{s_y}$, $I_{s_z}$,x_rot, y_rot, z_rot)
/* sort image space coordinates sets into back to front order, with deepest
   octant's coordinates in leading set position */
  front-to-back_sort({I})
/* Determine the contribution of each octant to the final image */
  projection_total = project_into_image_tree(I, image_rootptr, $x'_c$, $y'_c$, $z'_c$, 1)
/* recursively compute all image space coordinates */
  recurse_display(I, object_rootptr, image_rootptr, 1)
  }
```

```
recurse_display(I, objecttree_ptr, imagetree_ptr, recursion_level)
  {
  if recursion_level < log$_2$(N) then
  {

  for j = 0 to 7 do
  {
  k = map(j)
/* determine if the child octant is predicted to have spots to write to */
  if imagetree_ptr->child_ptrarray[j]->node_count > 0 then
  {
/* new total = starting parent total - (starting child total - ending child total) */
  imagetree_ptr->node_count = imagetree_ptr->node_count -
          (imagetree_ptr->child_ptrarray[j] - recurse_display(I,
                          objecttree_ptr->branch_array[k],
                          imagetree_ptr->child_ptrarray[j], recursion_level+1))
  } } /* close then and loop for j */
/* make sure that no holes remain within the space nominally filled-in by this octant
  by checking the node_count value, if there are gaps try the unexamined octants */
if imagetree_ptr->node_count ≠ 0 then
  {
/* perform transformation on the octants skipped at first, but must do all of them
  and all their sub-octants */
  for j = 0 to 7 do
  {
  k = map(j)
  image_ptr->node_count =
          exhaustive_recurse_display(I, objecttree_ptr->branch_array[k],
                          imagetree_ptr->child_ptrarray[j],recursion_level+1,
                              image_ptr->node_count)

  }}} /* close check for holes and check for recursion_level < log$_2$(N) */

if recursion_level = log$_2$(N) then
  {
/* generate the image tree and object tree pointers */
  for j = 0 to 7 do
  {
  k = map(j)
/* start by first doing the leaf octants predicted to have image space voxels which
  will be able to write to screen space */
  if (imagetree_ptr->child_ptrarray[j]->image_leaf.count > 0) then
  {
  imagetree_ptr->examined[j] = 1      /* make sure never do leaf again */
  if (objecttree_ptr->branch_array[k]->object_leaf ⊃ Window) then
  {
```

```
/* are at the deepest branch of the object tree, so to achieve supersampling just
   plot the eight image space points using the one required object space
   leaf's value */
       for h = 0 to 7 do
       {
       /* compute new image space octant center coordinates from stored values */
       if h = (1,3,5,7)
            then $C_x''$ = image_ptr->child_ptrarray[j]->image_leaf.x' + ($I_{h_x}$ >> m+1)
            else $C_x''$ = image_ptr->child_ptrarray[j]->image_leaf.x' - ($I_{h_x}$ >> m+1)
       if h = (2,3,6,7)
            then $C_y''$ = image_ptr->child_ptrarray[j]->image_leaf.y' + ($I_{h_y}$ >> m+1)
            else $C_y''$ = image_ptr->child_ptrarray[j]->image_leaf.y' - ($I_{h_y}$ >> rm+1)
       if h = (0,1,2,3)
            then $C_z''$ = image_ptr->child_ptrarray[j]->image_leaf.z' + ($I_{h_z}$ >> m+1)
            else $C_z''$ = image_ptr->child_ptrarray[j]->image_leaf.z' - ($I_{h_z}$ >> m+1)
       if (*behind*($C_x''$,$C_y''$,$C_y''$,Plane_Coeffs) and image_array[$C_x''$][$C_y''$] = -1) then
       {
       image_array[$C_x''$][$C_y''$] = objecttree_ptr->branch_array[k]->object_leaf
       image_ptr->node_count = image_ptr->node_count - 1
       projection_total = projection_total - 1;
         }
         }}} /* close for loop */
/* make sure that no holes remain which this octant could fill in */
   if image_ptr->node_count ≠ 0 then  /* transform any leaf nodes not plotted */
   {
/* generate the branch level image tree and object tree pointers for skipped octants */
   for j = 0 to 7 do
   {
   k = *map*(j)
/* examine those leaf octants not already transformed into image space */
   if image_ptr->examined[j] ≠ 1 then /* leaf not checked */
   {
   image_ptr->examined[j] = 1             /* don't check the leaf again */
/* are at the deepest branch of the object tree, so to achieve supersampling just plot
   the eight image space points using the one required object space leaf's value */
   if (objecttree_ptr->branch_array[k]->object_leaf $\supset$ Window) then
   {
       for h = 0 to 7 do
       {
       /* compute new image space octant center coordinates from stored values */
```

```
        if h = (1,3,5,7)
            then $C_x$'' = image_ptr->child_ptrarray[j]->image_leaf.x' + ($I_{h_x}$ >> m+1)
            else $C_x$'' = image_ptr->child_ptrarray[j]->image_leaf.x' - ($I_{h_x}$ >> m+1)
        if h = (2,3,6,7)
            then $C_y$'' = image_ptr->child_ptrarray[j]->image_leaf.y' + ($I_{h_y}$ >> m+1)
            else $C_y$'' = image_ptr->child_ptrarray[j]->image_leaf.y' - ($I_{h_y}$ >> rm+1)
        if h = (0,1,2,3)
            then $C_z$'' = image_ptr->child_ptrarray[j]->image_leaf.z' + ($I_{h_z}$ >> m+1)
            else $C_z$'' = image_ptr->child_ptrarray[j]->image_leaf.z' - ($I_{h_z}$ >> m+1)

        if (*behind*($C_x$'',$C_y$'',$C_y$'',Plane_Coeffs) and image_array[$C_x$''][$C_y$''] = -1) then
        {
        image_array[$C_x$''][$C_y$''] = objecttree_ptr->branch_array[k]->object_leaf
        image_ptr->node_count = image_ptr->node_count - 1
        projection_total = projection_total - 1;
          }}}
          }  /* close node_count check */
          }}  /* close outer for loop */
    }  /* close recursion level = $\log_2(N)$ */
  /* update the parent's total */
    if (image_ptr->node_count $\geq$ 0) then return(image_ptr->node_count)
            else return(0)
    }  /* close recurse display function */
```

```
exhaustive_recurse_display(I, objecttree_ptr, imagetree_ptr, recursion_level,total)
  {
  if recursion_level < log2(N) then
  {
  for j = 0 to 7 do
  {
  k = map(j)
/* write the octant and update this level's total to be written */
  total = exhaustive_recurse_display(I, objecttree_ptr->branch_array[k],
                              imagetree_ptr->child_ptrarray[j],
                              recursion_level+1, total)
/* can prematurely return only when no possibility of gaps exists */
  if (projection_total = 0) then return(0)
  }  /* close then and loop for j */
  } /* close < log2(N) */

  if recursion_level = log2(N) then
  {
/* generate the image tree and object tree pointers */
  for j = 0 to 7 do
  {
  k = map(j)
  if (image_ptr->examined[j] ≠ 1) then
  {
  image_ptr->examined[j] = 1          /* make sure leaf never done again */
/* are at the deepest branch of the object tree, so to achieve supersampling just
  plot the eight image space points using the one required object space
  leaf's value */
  if (objecttree_ptr->branch_array[k]->object_leaf ⊃ Window) then
     {
     for h = 0 to 7 do
     {
     /* compute new image space octant center coordinates from stored values */
     if h = (1,3,5,7)
          then Cx" = image_ptr->child_ptrarray[j]->image_leaf.x' + (Ihx >> m+1)
          else Cx" = image_ptr->child_ptrarray[j]->image_leaf.x' - (Ihx >> m+1)
     if h = (2,3,6,7)
          then Cy" = image_ptr->child_ptrarray[j]->image_leaf.y' + (Ihy >> m+1)
          else Cy" = image_ptr->child_ptrarray[j]->image_leaf.y' - (Ihy >> m+1)
     if h = (0,1,2,3)
          then Cz" = image_ptr->child_ptrarray[j]->image_leaf.z' + (Ihz >> m+1)
          else Cz" = image_ptr->child_ptrarray[j]->image_leaf.z' - (Ihz >> m+1)

     if (behind(Cx",Cy",Cy",Plane_Coeffs) and image_array[Cx"][Cy"] = -1) then
     {
     image_array[Cx"][Cy"] = objecttree_ptr->branch_array[k]->object_leaf
     projection_total = projection_total - 1;
     total = total - 1;
```

```
          }
           }} /* close for h loop and object value test */
   /* can only prematurely return when no possibility of gaps exists */
          if (projection_total = 0) then return(0)
           }} /* close for j loop */
   /* can't put parent total below zero, but make sure value propagates back to
   recurse_display so that calling level's parent total reflects holes filled in */
          if (total ≥ 0) then return(total) else return(0)
} /* close exhaustive_recurse_display */


recurse_build_objecttree($C_{o_x}$, $C_{o_y}$, $C_{o_z}$, objecttree_ptr,recursion_level)
{
/* compute the object space coordinates (which are identical to the object_array
               values) for each of the octants */
for k = 0 to 7 do
  {
  if k = (1,3,5,7)  then $C_x = C_{o_x}$ + ($O_{k_x}$ >> recursion_level[1])
               else $C_x = C_{o_x}$ - ($O_{k_x}$ >> recursion_level)
  if k = (2,3,6,7)  then $C_y = C_{o_y}$ + ($O_{k_y}$ >> recursion_level)
               else $C_y = C_{o_y}$ - ($O_{k_y}$ >> recursion_level)
  if k = (0,1,2,3)  then $C_z = C_{o_z}$ + ($O_{k_z}$ >> recursion_level)
                   else $C_z = C_{o_z}$ - ($O_{k_z}$ >> recursion_level)

/* if are at a level in the oct-tree where branch nodes are required, create a new
  branch node in the object space oct-tree holding an array of eight pointers */
  if recursion_level < $\log_2(N)$ then
                 objecttree_ptr->branch_array[k]->new_objecttree_node
/* if are at a level in the oct-tree where leaf nodes are required, create a new leaf
  node in the object space oct-tree holding an integer density value */
  if recursion_level = $\log_2(N)$ then
                 objecttree_ptr->branch_array[k]->object_leaf

         if recursion_level < $\log_2(N)$
                   then    /* pass coordinates and pointer to next level in oct-tree */
            recurse_build_objecttree($C_x$, $C_y$, $C_z$, objecttree_ptr->branch_array[k],
                                         recursion_level+1)

  if recursion_level = $\log_2(N)$  /* record the object array value in the oct-tree leaf */
                   then
                   objecttree_ptr->branch_array[k]-> object_leaf  =
                                                      object_array[$C_x$][$C_y$][$C_z$]

}}
```

[1]>> is defined to be the right-shift operator.

```
*project_into_image_tree* (I,image_ptr, $C_{ix}$, $C_{iy}$, $C_{iz}$, recursion_level)
{
image_ptr->node_count = 0
for j=0 to 7 do
  {
/* compute new image space octant center coordinates */
  if j = (1,3,5,7)   then $C_x' = C_{ix} + (I_{jx}$ >> recursion_level)
            else $C_x' = C_{ix} - (I_{jx}$ >> recursion_level)
  if j = (2,3,6,7)   then $C_y' = C_{ix} + (I_{jy}$ >> recursion_level)
            else $C_y' = C_{ix} - (I_{jy}$ >> recursion_level)
  if j = (0,1,2,3)   then $C_z' = C_{ix} + (I_{jz}$ >> recursion_level)
            else $C_z' = C_{ix} - (I_{jz}$ >> recursion_level)
  if recursion_level < $\log_2(N)$ then
            image_ptr->node_count = image_ptr->node_count +
            *project_into_image_tree*(I, image_ptr->child_ptrarray[j],
                                    x', y', z', recursion_level+1)
  if recursion_level = $\log_2(N)$ then
  {
  image_ptr->examined[j] = 0
  image_ptr->child_ptrarray[j]->image_leaf.count = 0 /* initialize total for leaf */
/* record the coordinates for later use in *recursive_display* */
  image_ptr->child_ptrarray[j]->image_leaf.x' = $C_x'$
  image_ptr->child_ptrarray[j]->image_leaf.y' = $C_y'$
  image_ptr->child_ptrarray[j]->image_leaf.z' = $C_z'$
/* compute the total number of points which can be written and retain the value */
  for k=0 to 7 do
  {
/* compute terminal image space octant center coordinates */
  if k = (1,3,5,7)  then $C_x'' = C_x' + (I_{kx}$ >> m+1)
            else $C_x'' = C_x' - (I_{kx}$ >> m+1)
  if k = (2,3,6,7)  then $C_y'' = C_y' + (I_{ky}$ >> m+1)
            else $C_y'' = C_y' - (I_{ky}$ >> m+1)
  if k = (0,1,2,3)  then $C_z'' = C_z' + (I_{kz}$ >> m+1)
            else $C_z'' = C_z' - (I_{kz}$ >> m+1)
/* now check to see if the image space voxel can be written */
  if *behind*($C_x''$, $C_y''$, $C_z''$, Plane_Coeffs) and update_box = -1 then
  {
/* so update counters and make sure no other image voxel can write to that entry */
  update_box[$C_x''$][$C_y''$] = 1
  image_ptr->node_count = image_ptr->node_count + 1
  image_ptr->child_ptrarray[j]->image_leaf.count =
                image_ptr->child_ptrarray[j]->image_leaf.count + 1
  }} } /* close for k=0 to 7  and = $\log_2 N$*/
  } /* close for j */
  return(image_ptr->node_count)  /* make sure the total is propagated up the tree */
} /* close function */
```

APPENDIX K

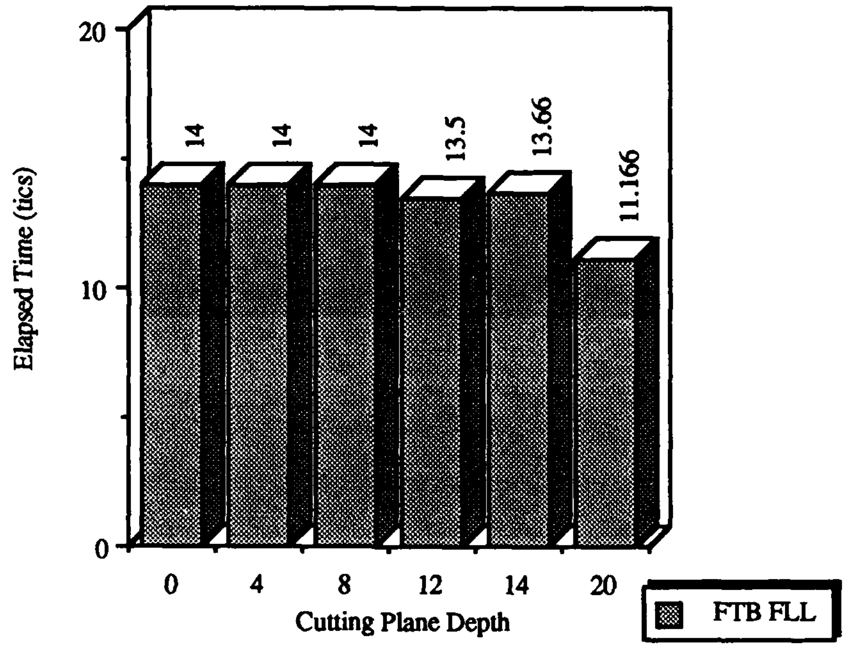
APPENDIX K
PERFORMANCE OF THE ADAPTIVE RECURSIVE FRONT-TO-BACK HIDDEN-SURFACE REMOVAL ALGORITHMS

K.1 INTRODUCTION.

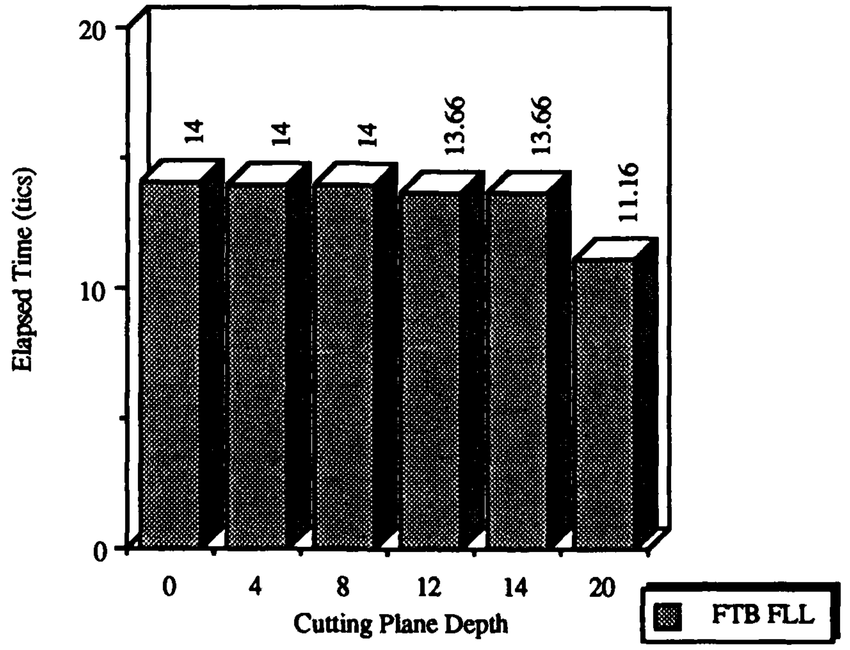
The performance results presented in this appendix depict the performance of the adaptive recursive front-to-back (FTB) hidden-surface removal (HSR) algorithm on a range of scene sizes and threshold window saturation values. The graphs in this appendix also compare the performance of the adaptive FTB HSR algorithm with the editing recursive HSR algorithm and the basic recursive HSR algorithm. For each cube (scene) size, the performance of the adaptive FTB hidden-surface removal algorithm using either front-lower left coordinates (FLL) or subscene center coordinates (SCC) as the $\{\mathcal{P}_0\}$ were tested. The presentation is organized with the algorithm's performance on 8^3, 16^3, and 32^3 scene sizes across a range of threshold window saturation values shown first to highlight the change in execution speed of each variant of the algorithm as cutting plane depth varies, with the results for each combination of scene size, σ, and algorithm variant presented in a series of graphs. This is followed by a series of graphs which depict the relative performance of the adaptive, basic, and editing FTB HSR algorithms across the same range of scene sizes. In both sequences of graphs, the FLL variant's results are presented first followed by the SCC variant

K.2 ADAPTIVE RECURSIVE FTB FLL HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN AN 8 X 8 X 8 CUBE.

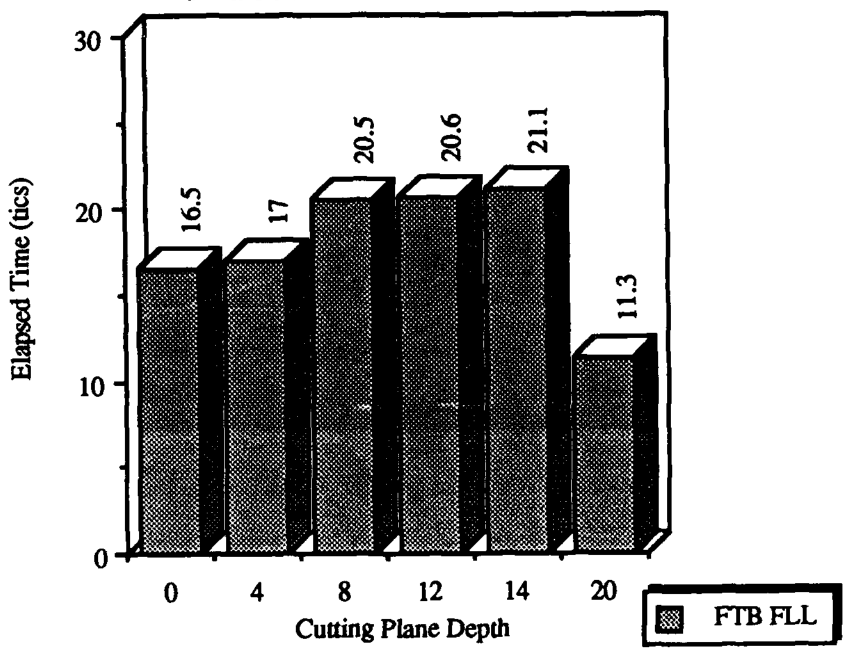
Adaptive Recursive Front-to-Back FLL HSR Algorithm Performance On a 100% Full 8 x 8 x 8 Cube



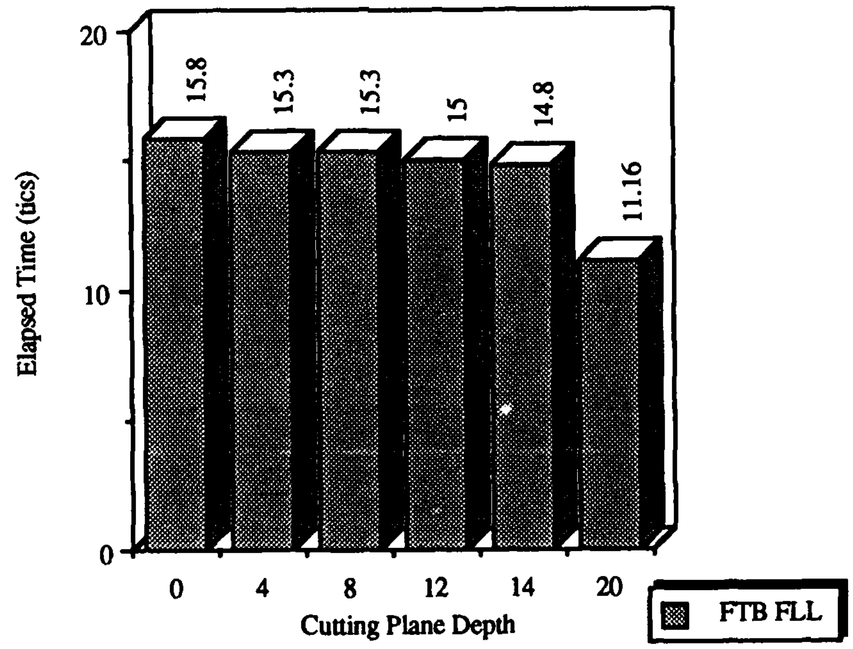
Adaptive Recursive Front-to-Back FLL HSR Algorithm Performance On a 90% Full 8 x 8 x 8 Cube



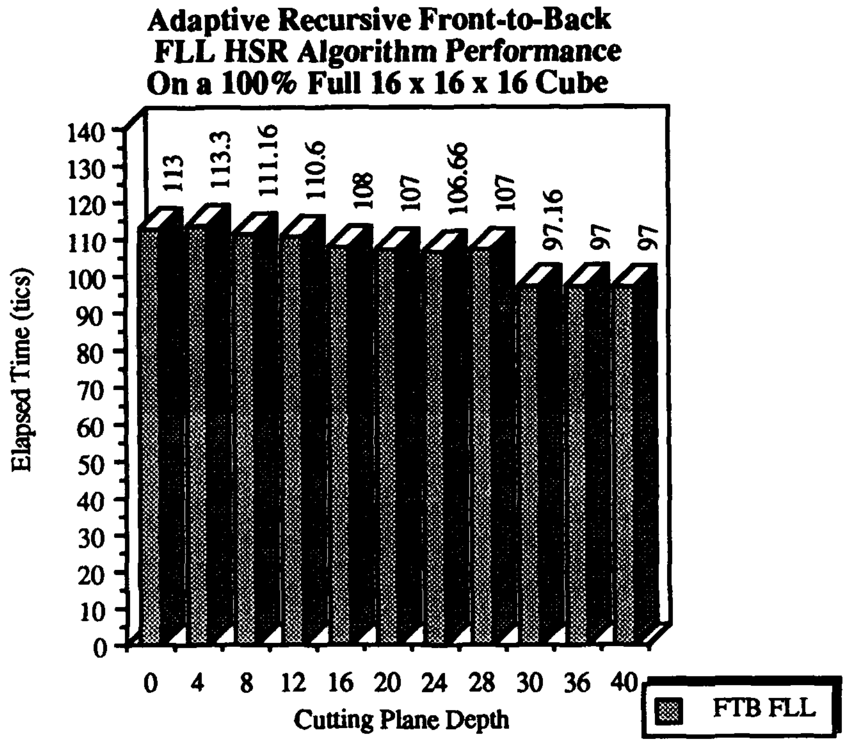
Adaptive Recursive Front-to-Back FLL HSR Algorithm Performance On a 50% Full 8 x 8 x 8 Cube

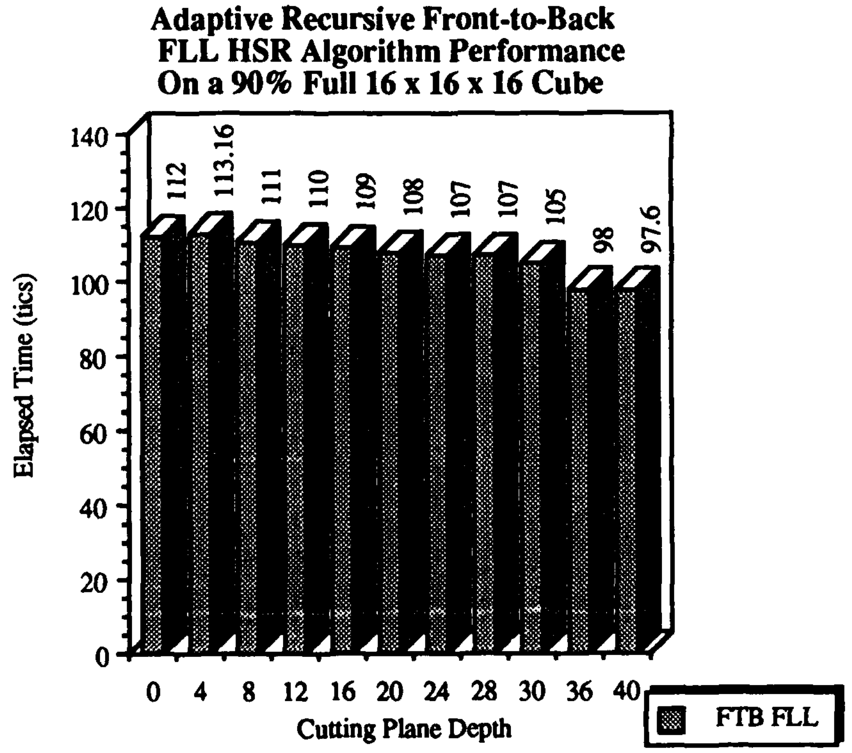


Adaptive Recursive Front-to-Back FLL HSR Algorithm Performance On a 10% Full 8 x 8 x 8 Cube

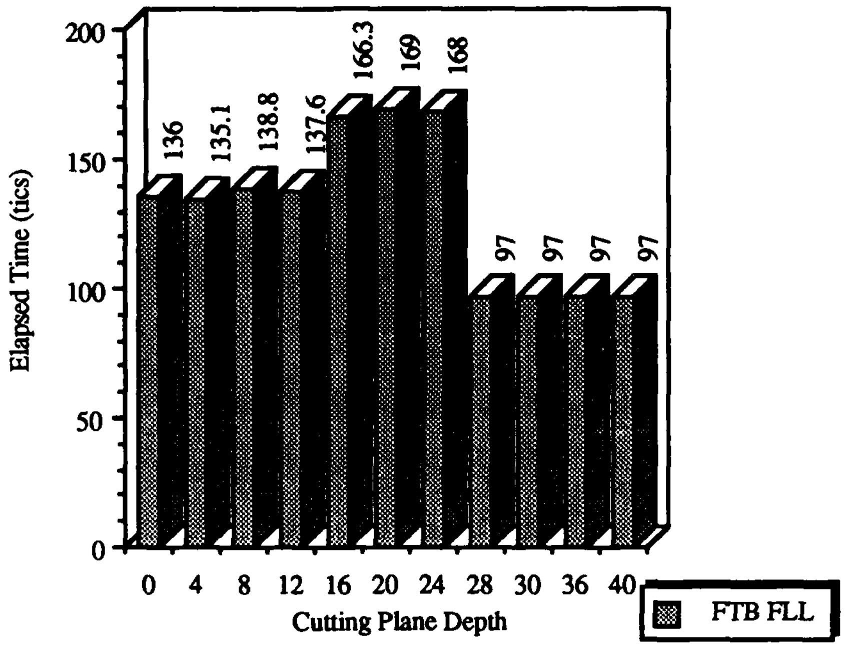


K.3 ADAPTIVE RECURSIVE FTB FLL HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN A 16 X 16 X 16 CUBE.

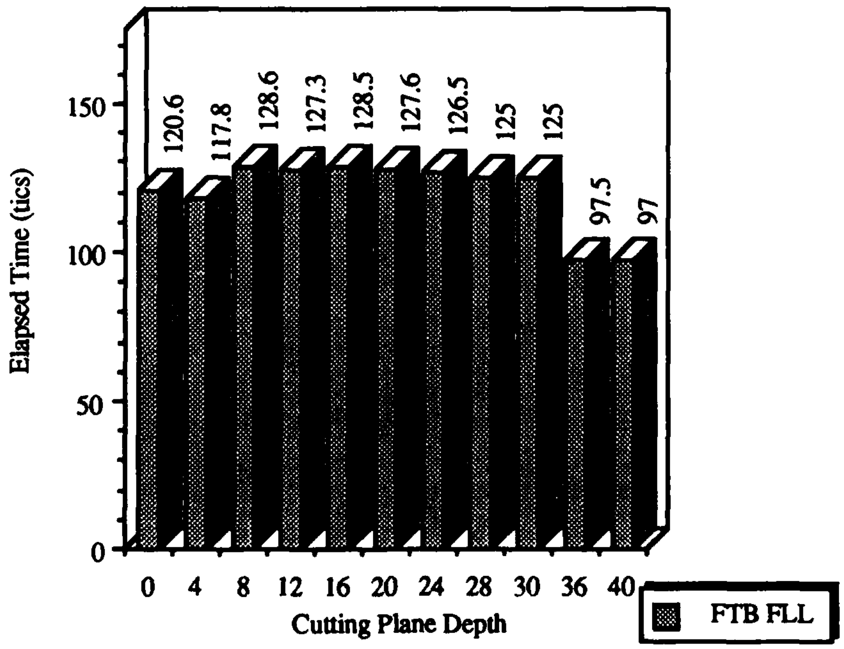




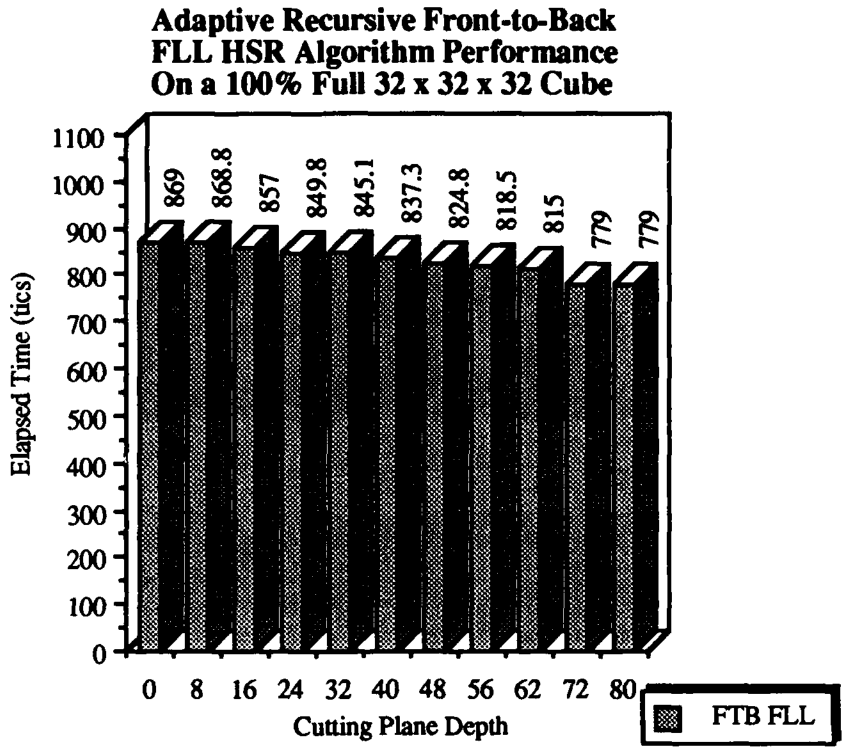
Adaptive Recursive Front-to-Back FLL HSR Algorithm Performance On a 50% Full 16 x 16 x 16 Cube

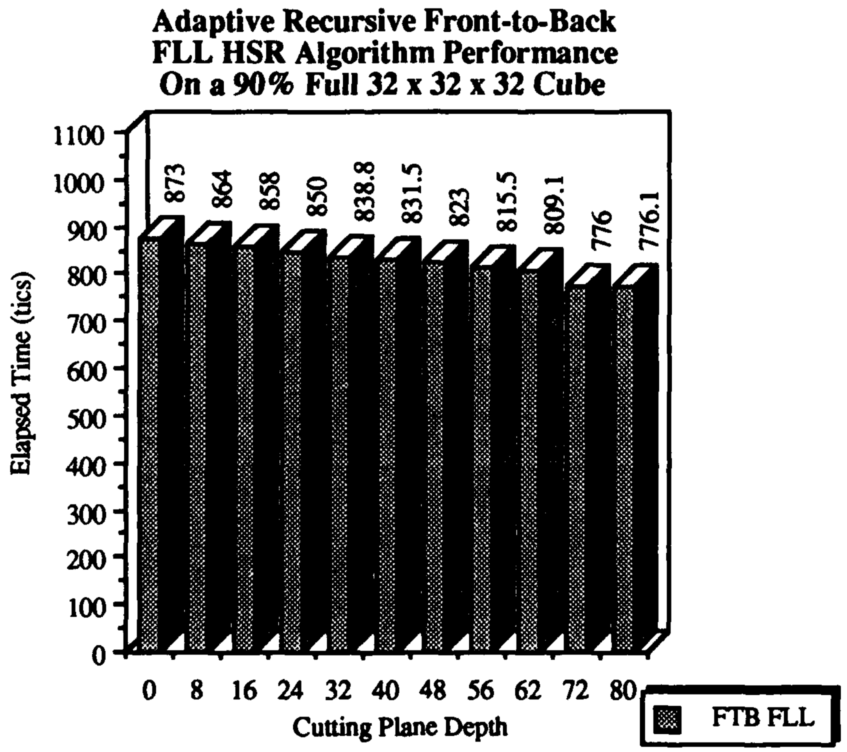


Adaptive Recursive Front-to-Back FLL HSR Algorithm Performance On a 10% Full 16 x 16 x 16 Cube

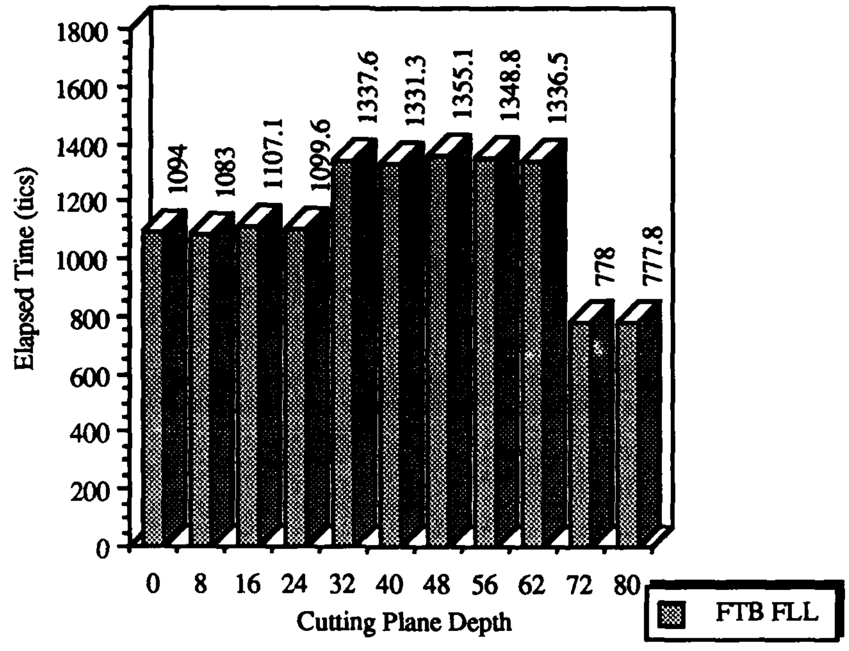


K.4 ADAPTIVE RECURSIVE FTB FLL HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN A 32 X 32 X 32 CUBE.

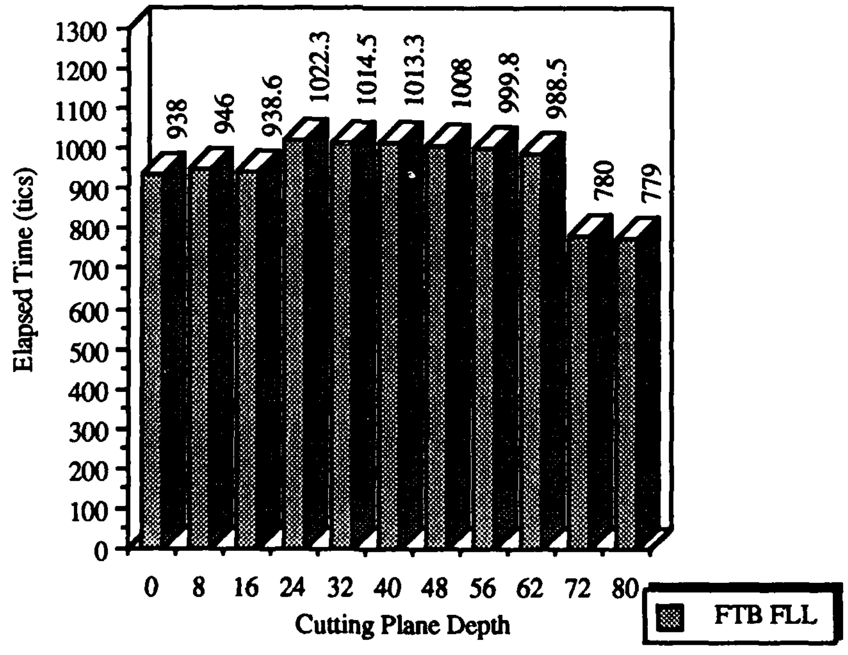




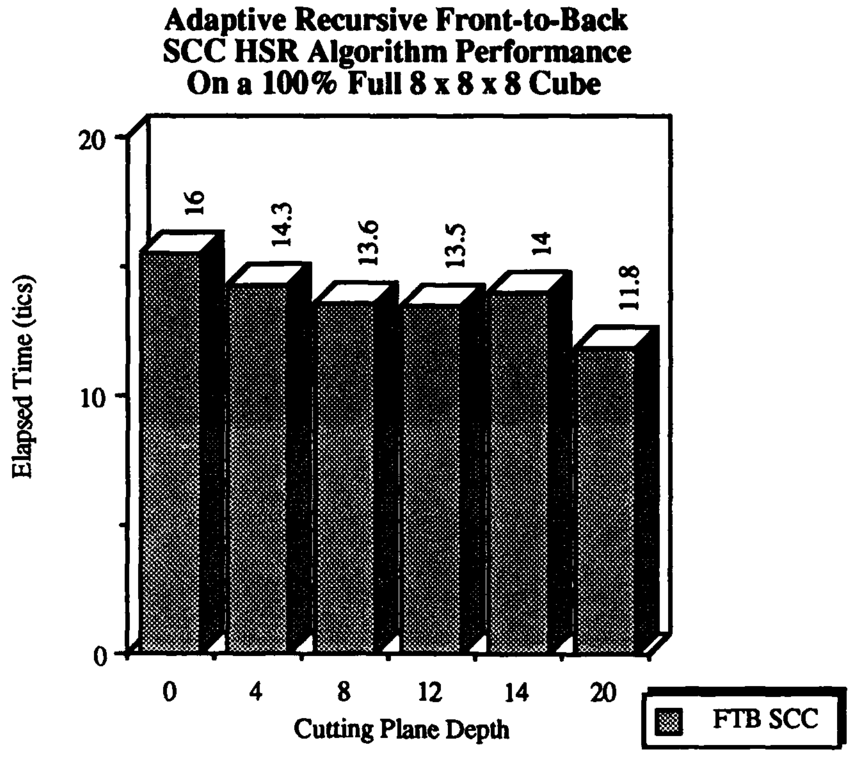
Adaptive Recursive Front-to-Back FLL HSR Algorithm Performance On a 50% Full 32 x 32 x 32 Cube

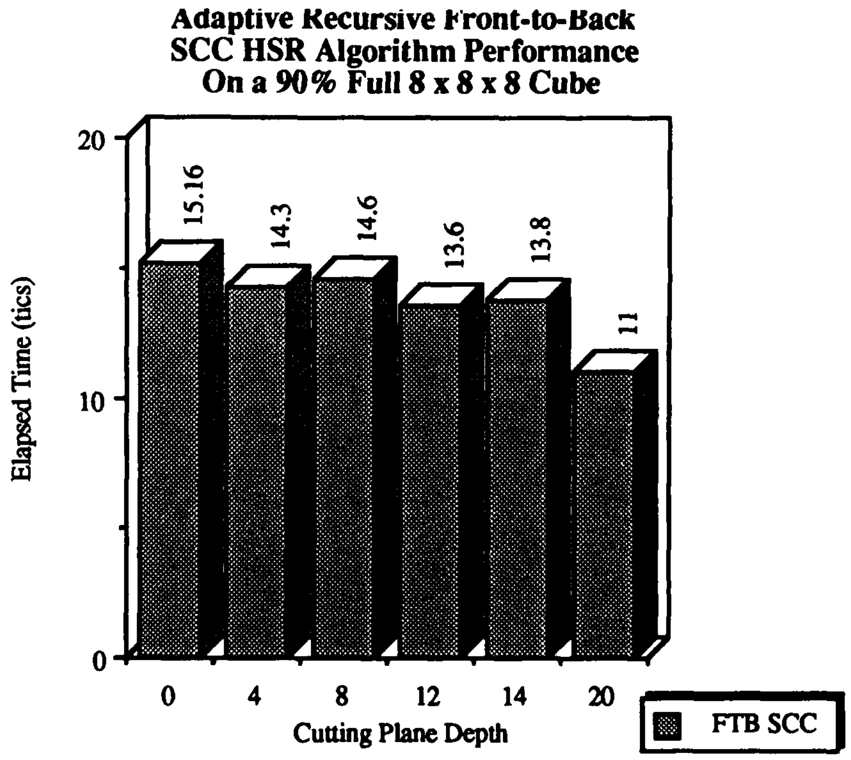


Adaptive Recursive Front-to-Back FLL HSR Algorithm Performance On a 10% Full 32 x 32 x 32 Cube

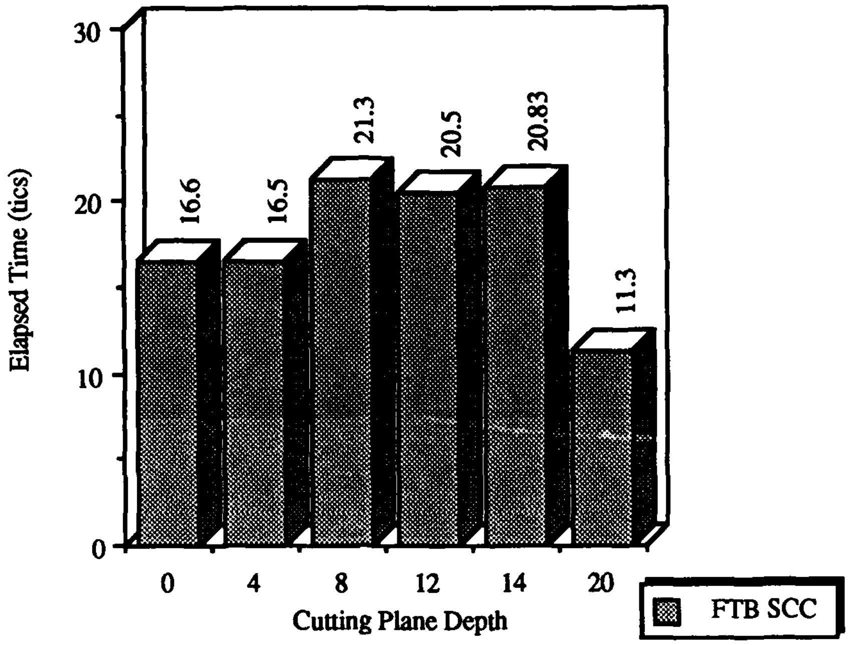


K.5 ADAPTIVE RECURSIVE FTB SCC HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN AN 8 X 8 X 8 CUBE.

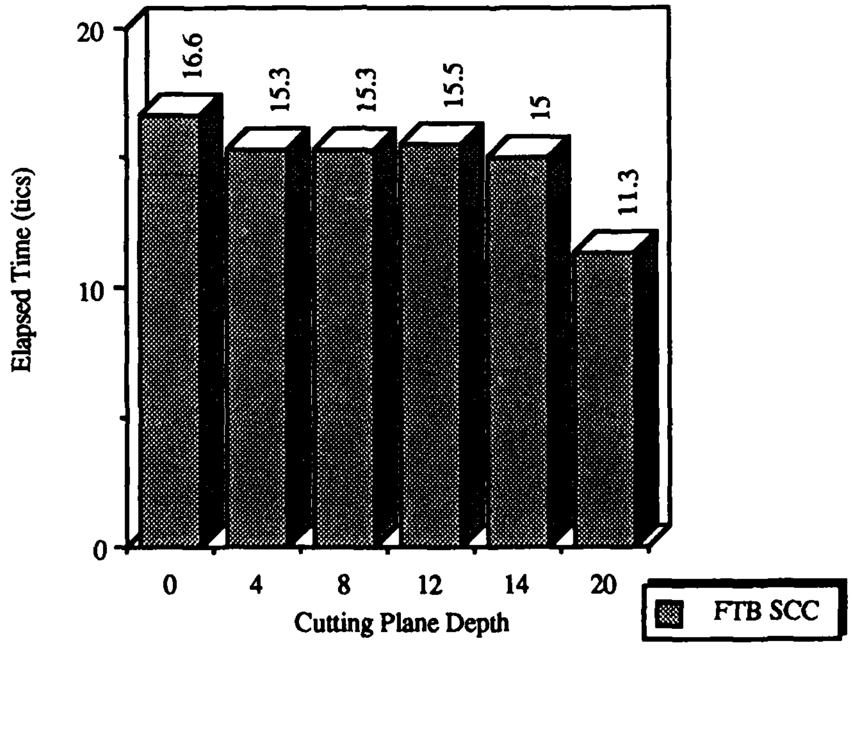




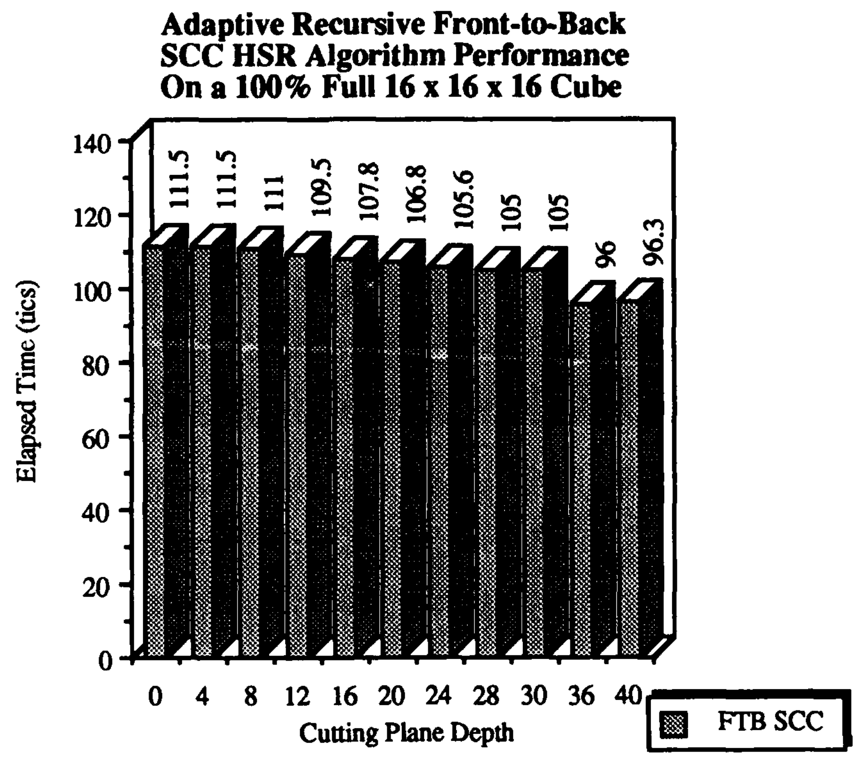
Adaptive Recursive Front-to-Back SCC HSR Algorithm Performance On a 50% Full 8 x 8 x 8 Cube

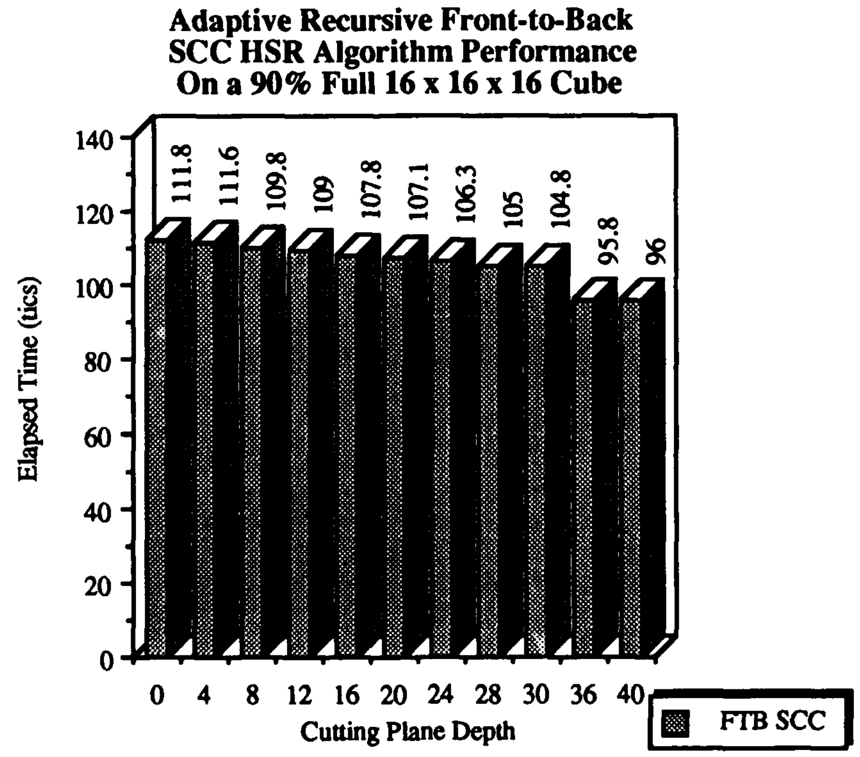


Adaptive Recursive Front-to-Back SCC HSR Algorithm Performance On a 10% Full 8 x 8 x 8 Cube

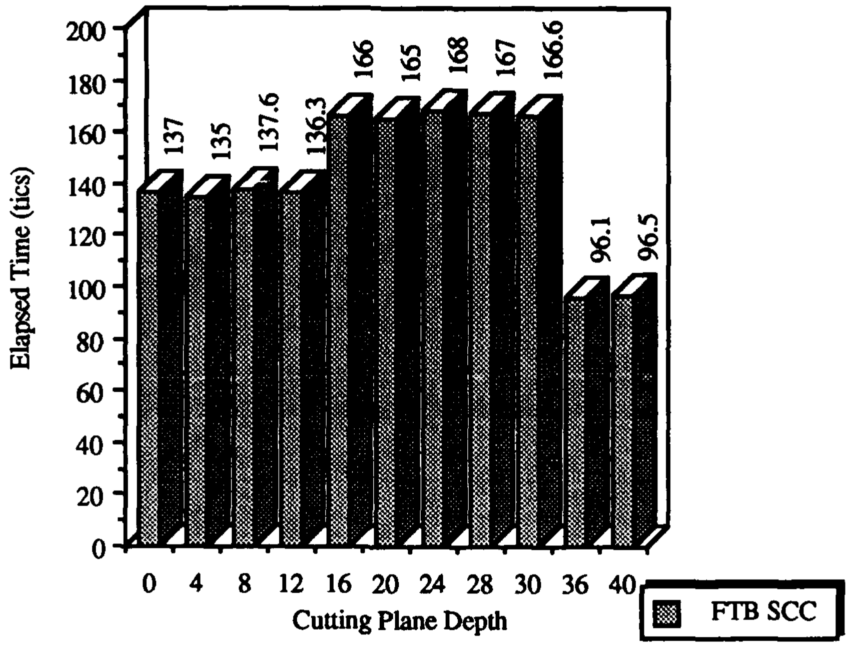


K.6 ADAPTIVE RECURSIVE FTB SCC HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN A 16 X 16 X 16 CUBE.

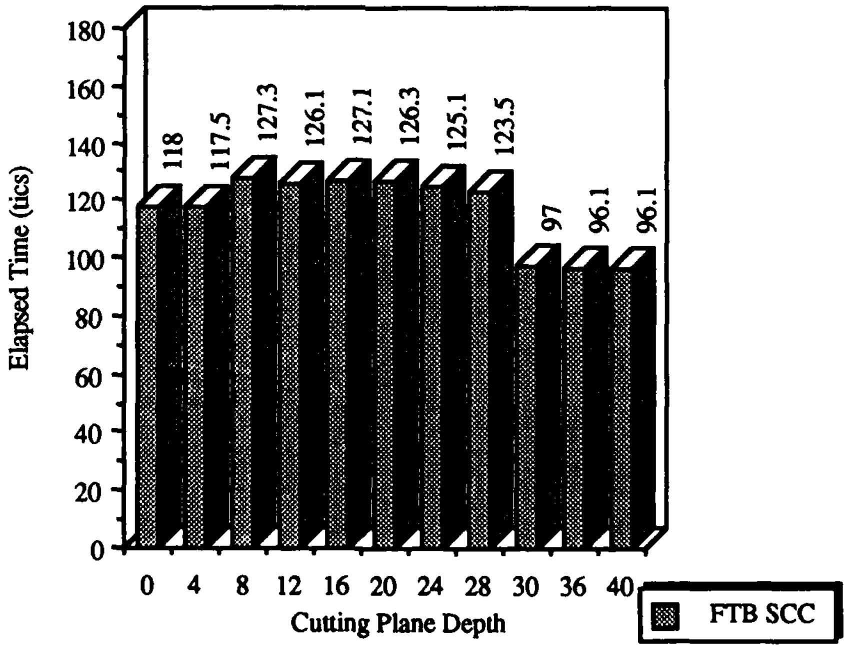




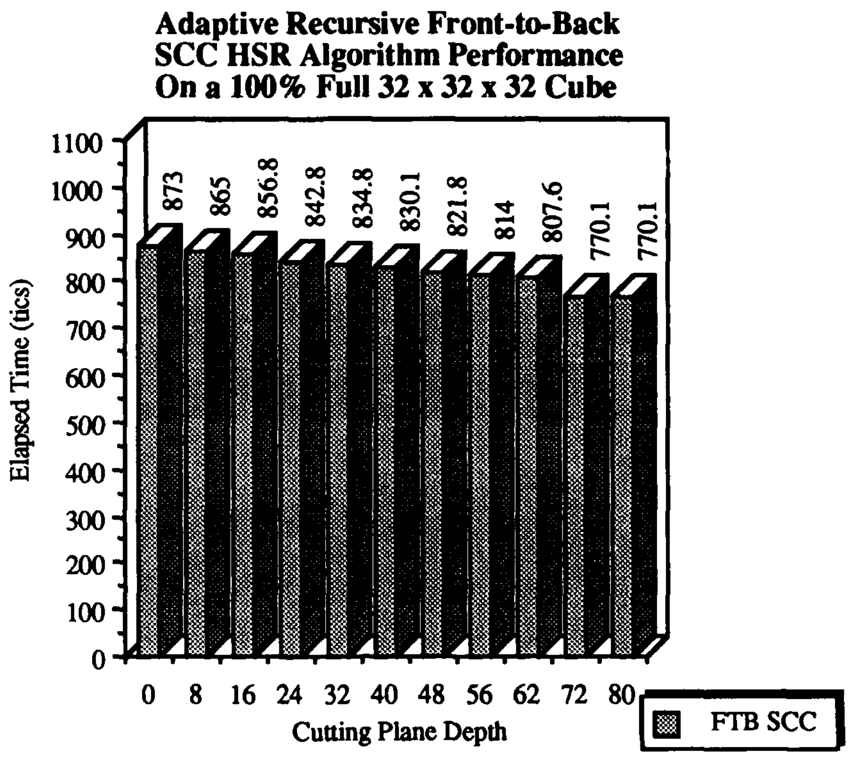
Adaptive Recursive Front-to-Back SCC HSR Algorithm Performance On a 50% Full 16 x 16 x 16 Cube

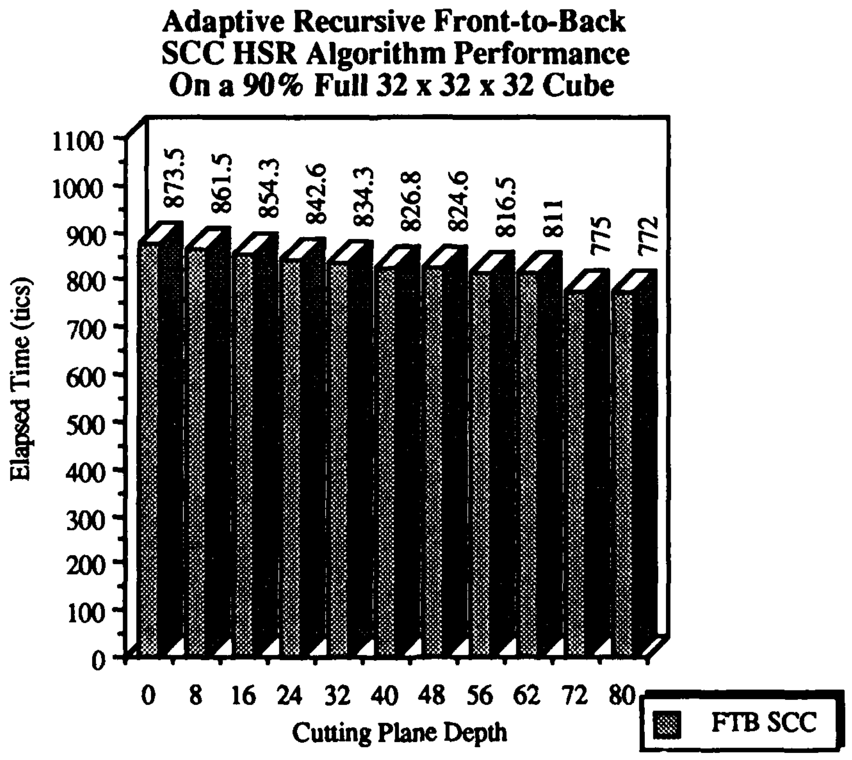


Adaptive Recursive Front-to-Back SCC HSR Algorithm Performance On a 10% Full 16 x 16 x 16 Cube

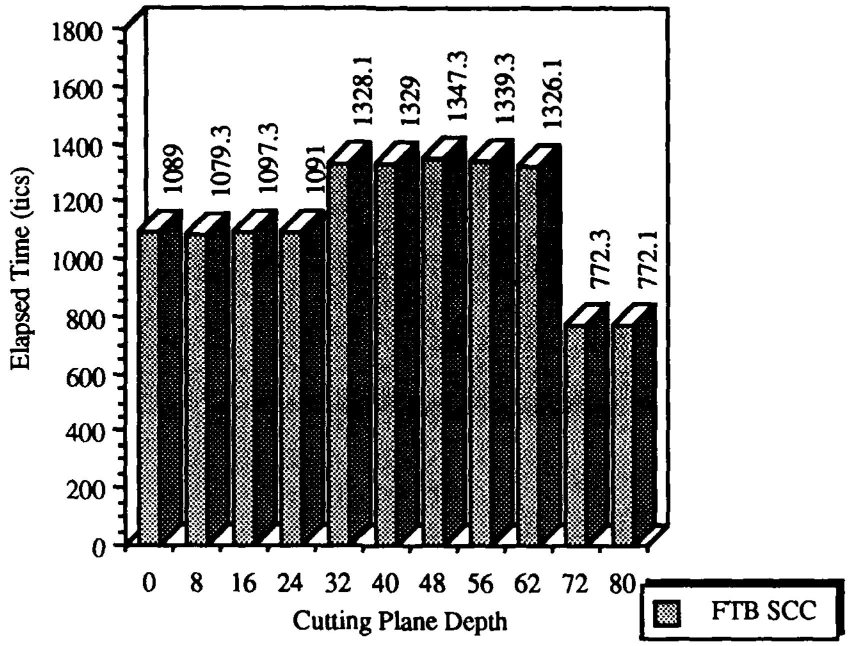


K.7 ADAPTIVE RECURSIVE FTB SCC HSR ALGORITHM PERFORMANCE FOR DIFFERENT σ VALUES IN A 32 X 32 X 32 CUBE.

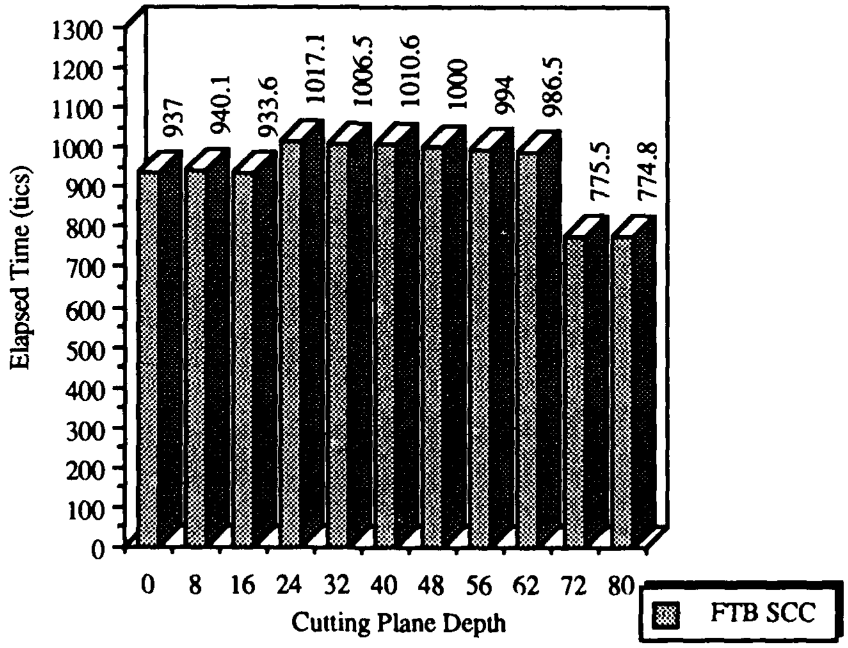




Adaptive Recursive Front-to-Back SCC HSR Algorithm Performance On a 50% Full 32 x 32 x 32 Cube

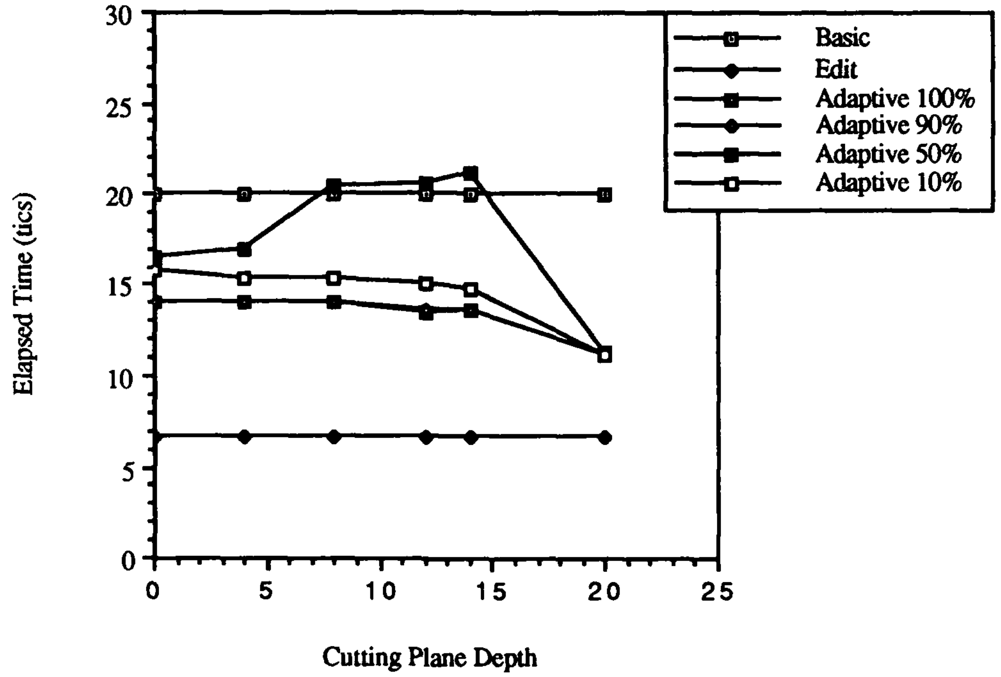


Adaptive Recursive Front-to-Back SCC HSR Algorithm Performance On a 10% Full 32 x 32 x 32 Cube

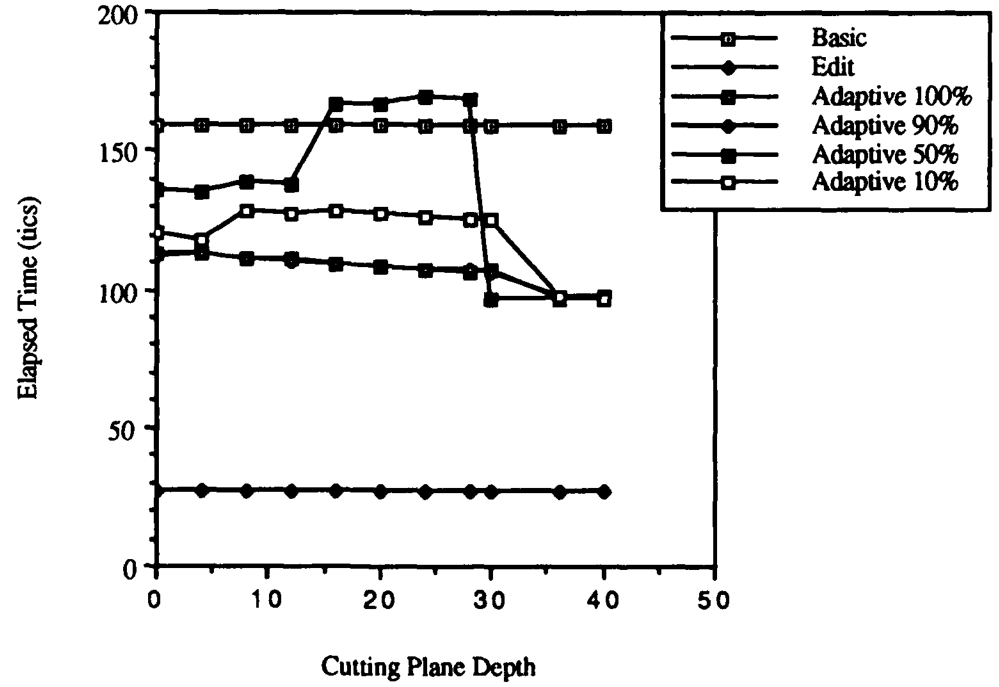


K.8 ADAPTIVE RECURSIVE FTB HSR ALGORITHM PERFORMANCE COMPARED TO EDITING AND BASIC FTB HSR ALGORITHMS.

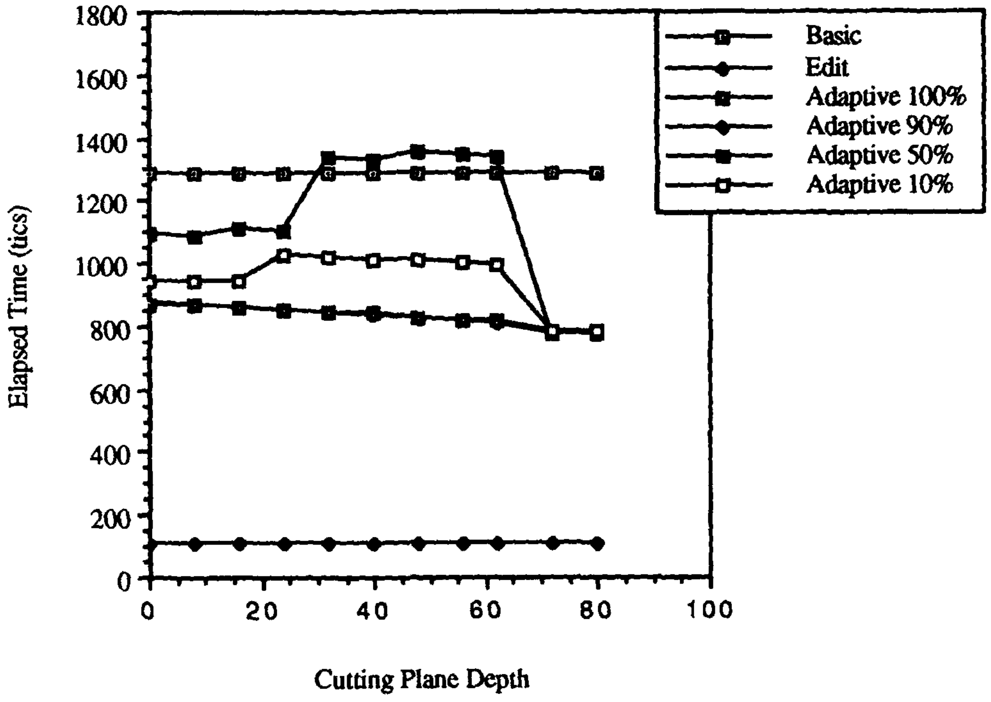
Comparison of Basic, Edit, and Adaptive FTB FLL HSR Algorithms On an 8 x 8 x 8 Cube at Different Thresholds



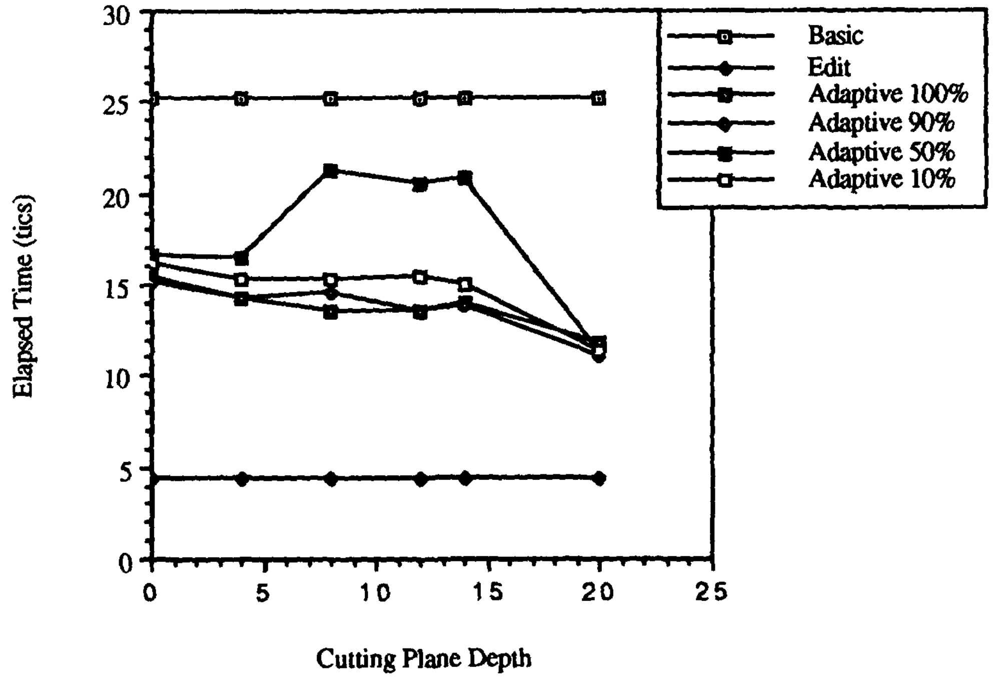
Comparison of Basic, Edit, and Adaptive FTB FLL HSR Algorithms On a 16 x 16 x 16 Cube at Different Thresholds



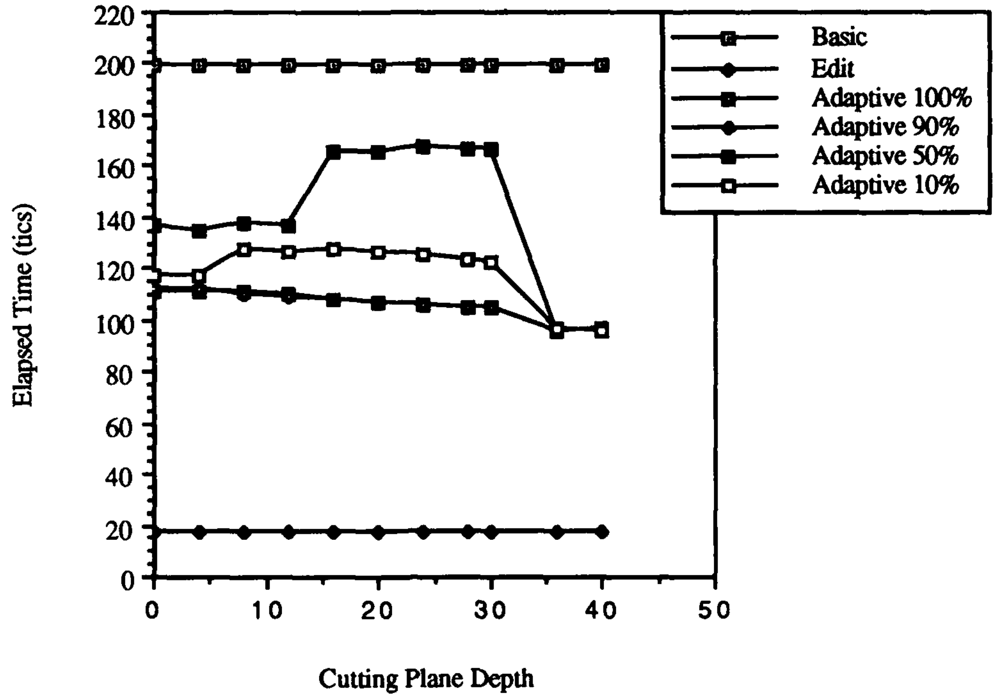
Comparison of Basic, Edit, and Adaptive FTB FLL HSR Algorithms On a 32 x 32 x 32 Cube at Different Thresholds



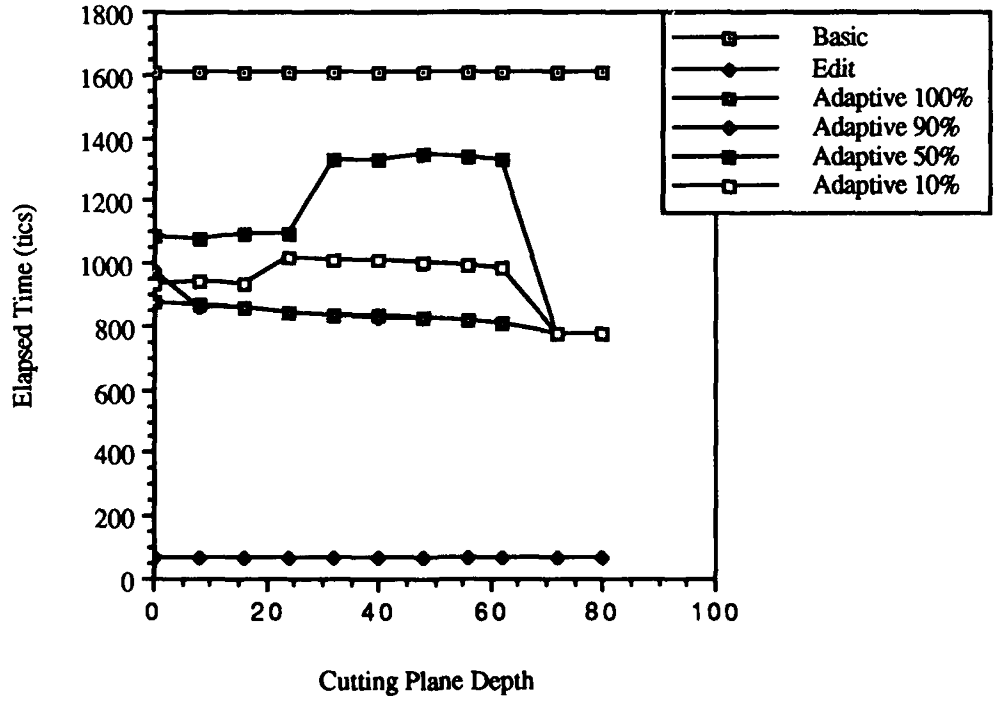
Comparison of Basic, Edit, and Adaptive FTB SCC HSR Algorithms On an 8 x 8 x 8 Cube at Different Thresholds



Comparison of Basic, Edit, and Adaptive FTB SCC HSR Algorithms On a 16 x 16 x 16 Cube at Different Thresholds



Comparison of Basic, Edit, and Adaptive FTB SCC HSR Algorithms On a 32 x 32 x 32 Cube at Different Thresholds



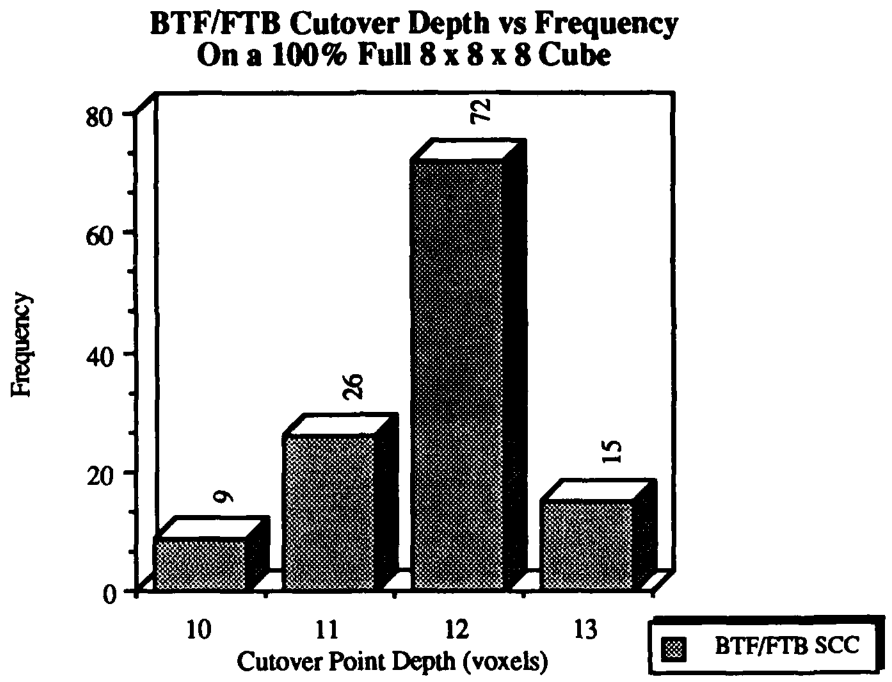
APPENDIX L

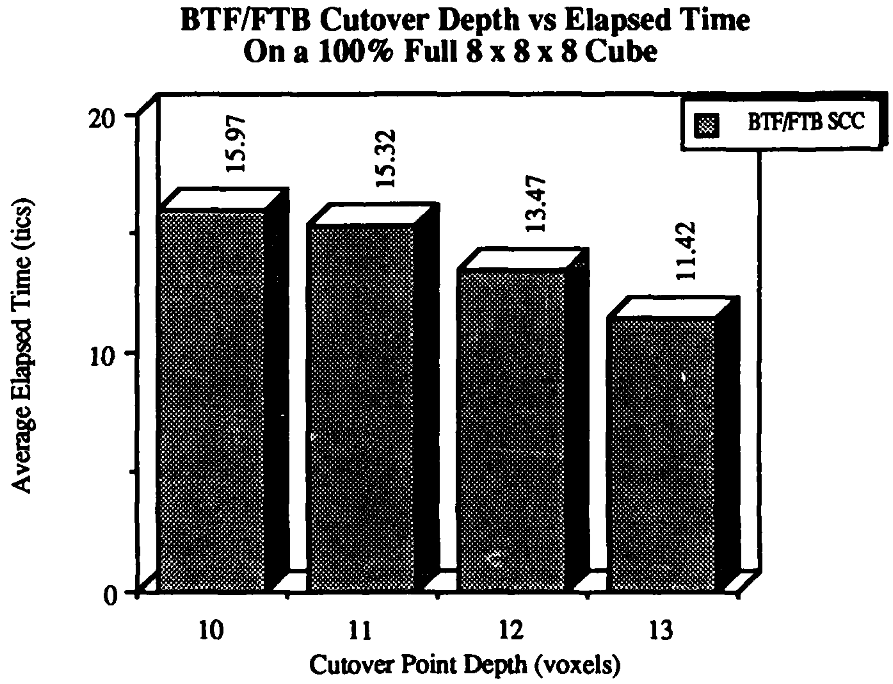
APPENDIX L
DATA USED IN ANALYSIS AND DESIGN OF THE DYNAMIC ADAPTIVE RECURSIVE HIDDEN-SURFACE REMOVAL ALGORITHM

L.1 INTRODUCTION.

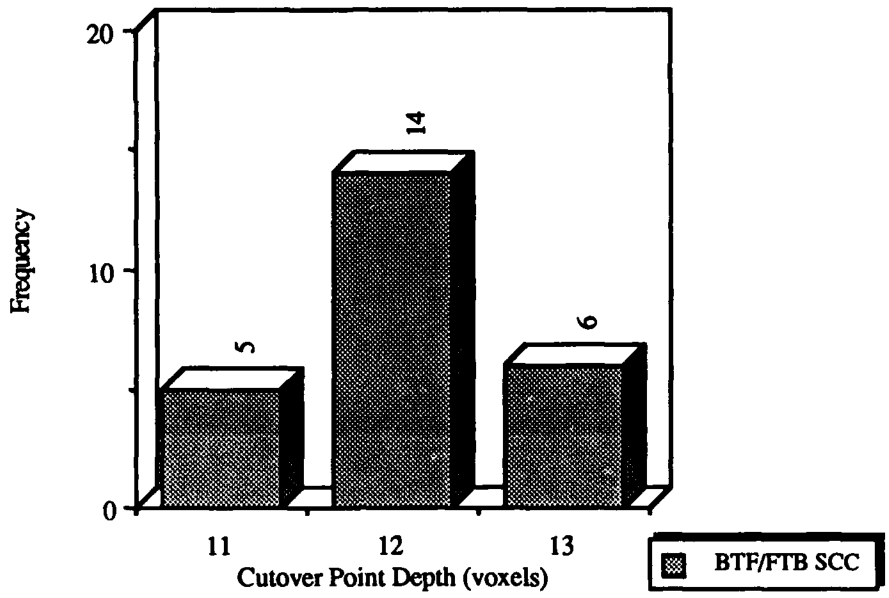
The results presented in this appendix depict the performance of the component BTF and FTB code which make up the dynamic adaptive recursive SCC-based hidden-surface removal (HSR) algorithm. The data gathered for the analysis presented in Chapter VI was gathered over a range of scene sizes and threshold window saturation values in order to better characterize the performance of the component algorithms. The presentation is organized based upon the object space volume and threshold window saturation value. For each $\{\mathcal{N}_o\}$ size (8^3, 16^3, and 32^3) σ was set to 10%, 50%, 90%, and 100% and the resulting cutting plane location for a set of x-axis and y-axis rotations was determined. These outcomes are presented in two graphs for each scene size and σ combination. The frequency of the cutting plane occurring at a given voxel depth for the given σ and object space volume is presented first, followed by the average elapsed image rendering time at the cutover point(s) for the σ/object space volume combination.

L.2. DYNAMIC ADAPTIVE RECURSIVE HSR ALGORITHM PERFORMANCE ON AN 8 X 8 X 8 CUBE.

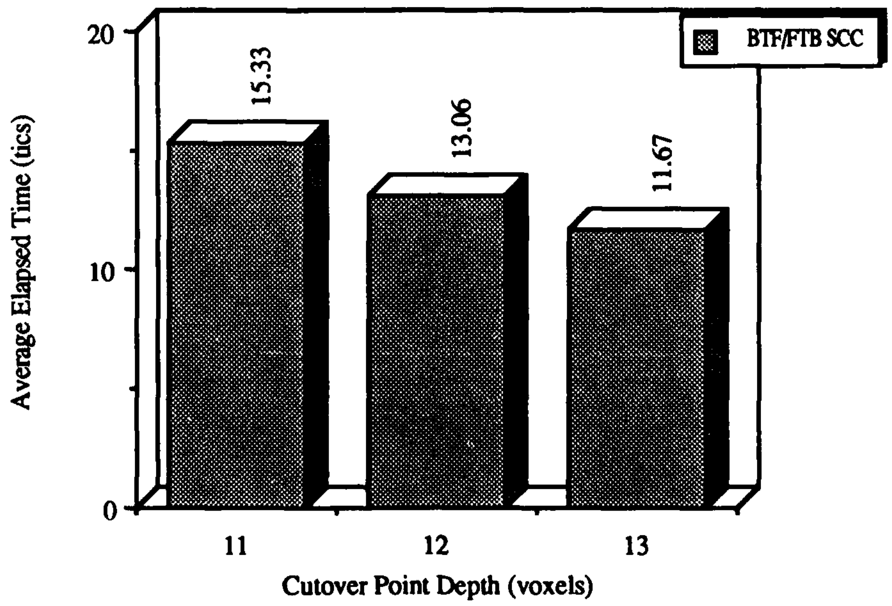




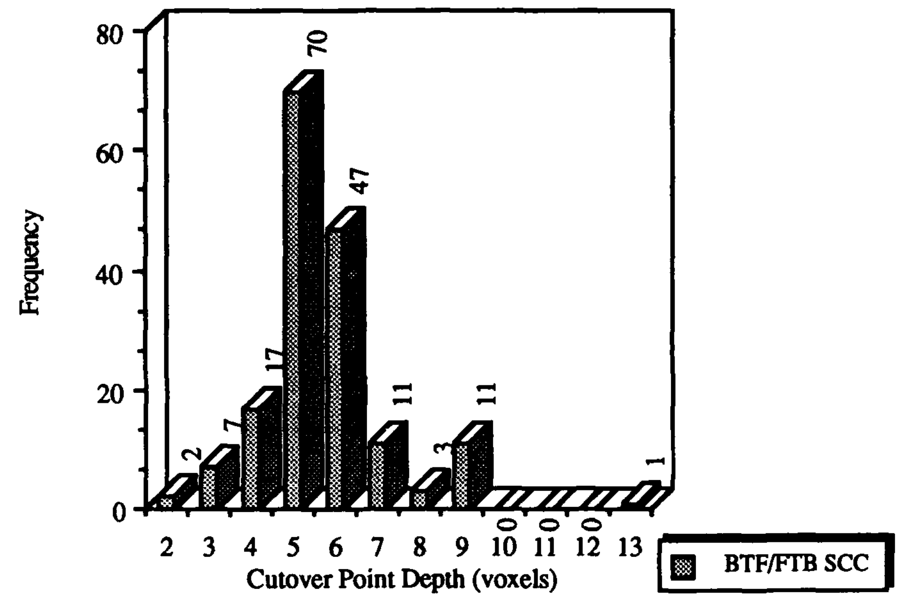
BTF/FTB Cutover Depth vs Frequency
On a 90% Full 8 x 8 x 8 Cube



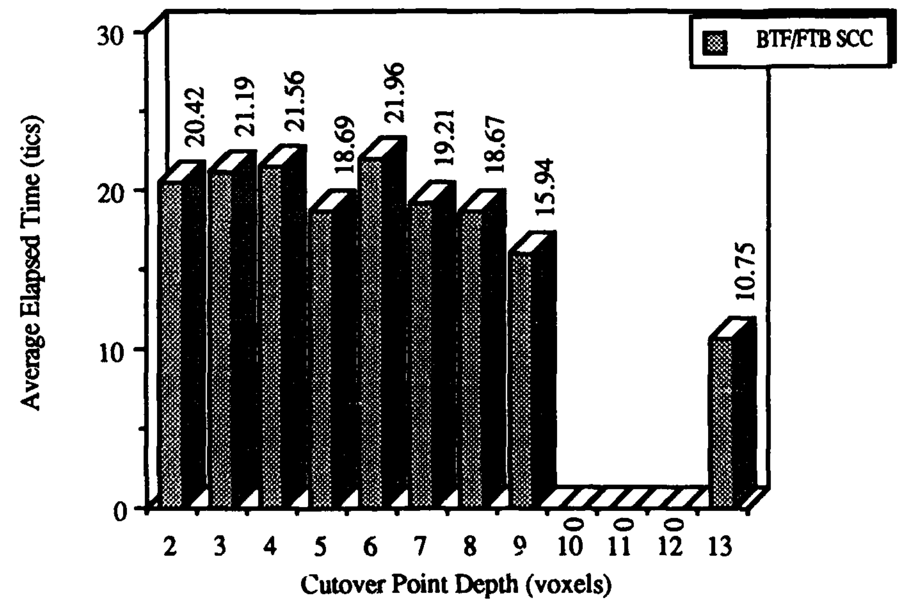
BTF/FTB Cutover Depth vs Elapsed Time
On a 90% Full 8 x 8 x 8 Cube



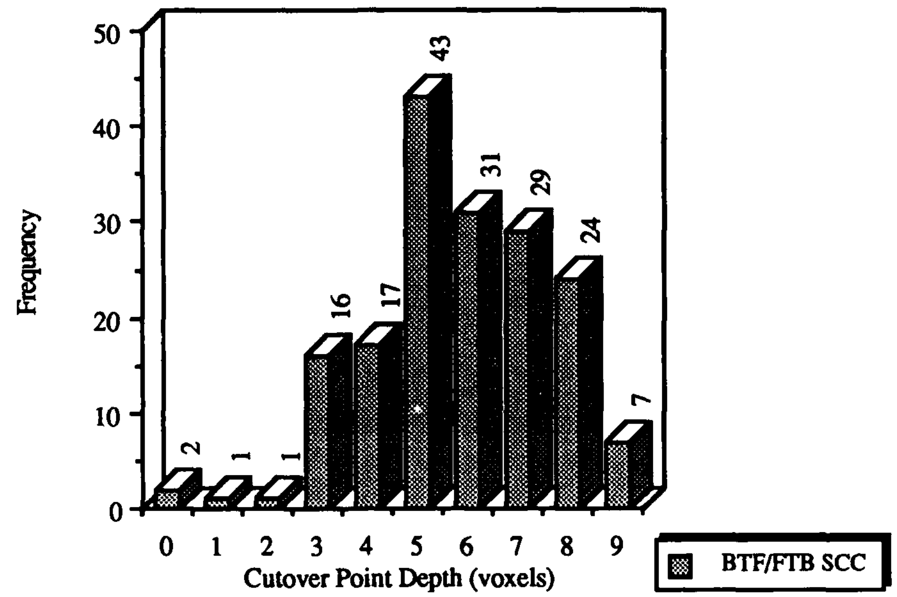
BTF/FTB Cutover Depth vs Frequency
On a 50% Full 8 x 8 x 8 Cube



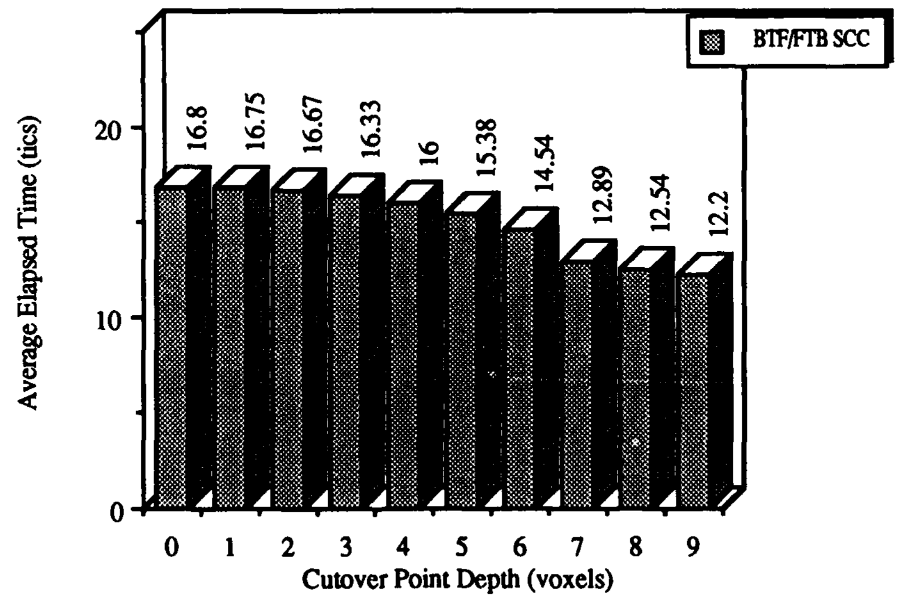
BTF/FTB Cutover Depth vs Elapsed Time
On a 50% Full 8 x 8 x 8 Cube



BTF/FTB Cutover Depth vs Frequency
On a 10% Full 8 x 8 x 8 Cube

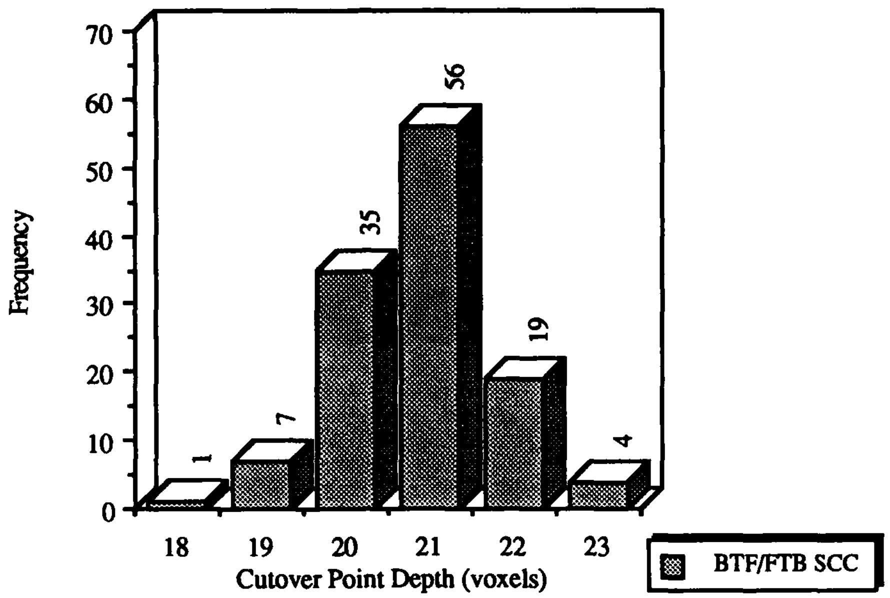


BTF/FTB Cutover Depth vs Elapsed Time
On a 10% Full 8 x 8 x 8 Cube

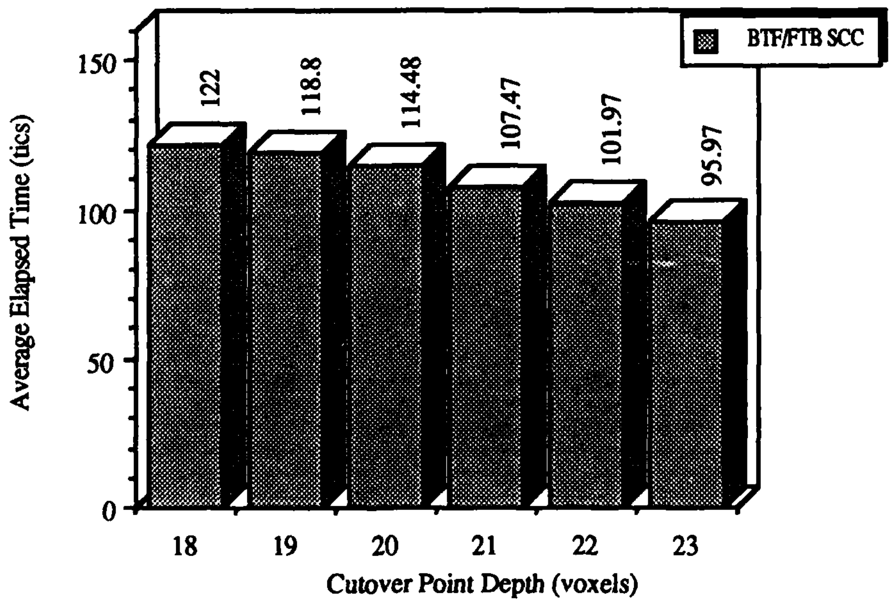


L.3. DYNAMIC ADAPTIVE RECURSIVE HSR ALGORITHM PERFORMANCE ON A 16 X 16 X 16 CUBE.

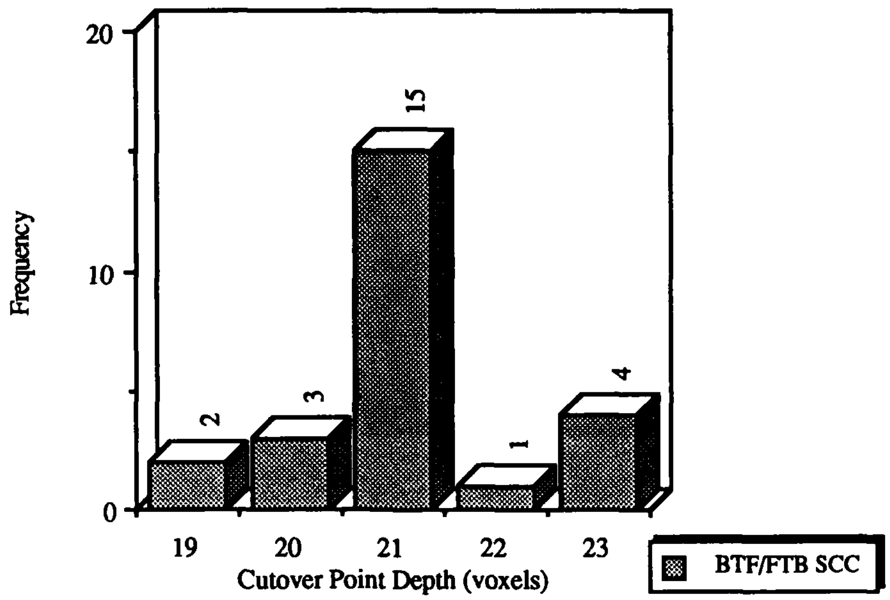
BTF/FTB Cutover Depth vs Frequency
On a 100% Full 16 x 16 x 16 Cube



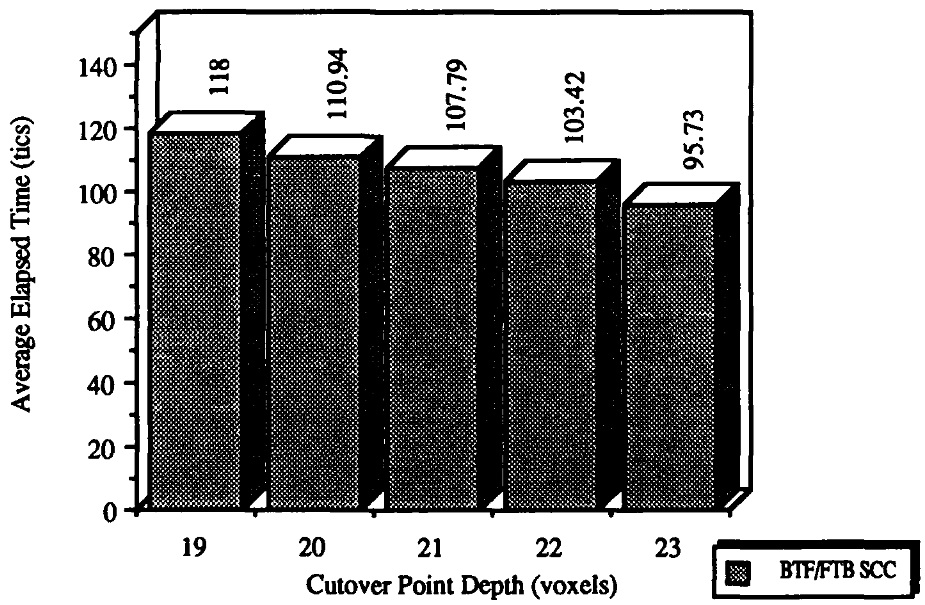
BTF/FTB Cutover Depth vs Elapsed Time
On a 100% Full 16 x 16 x 16 Cube



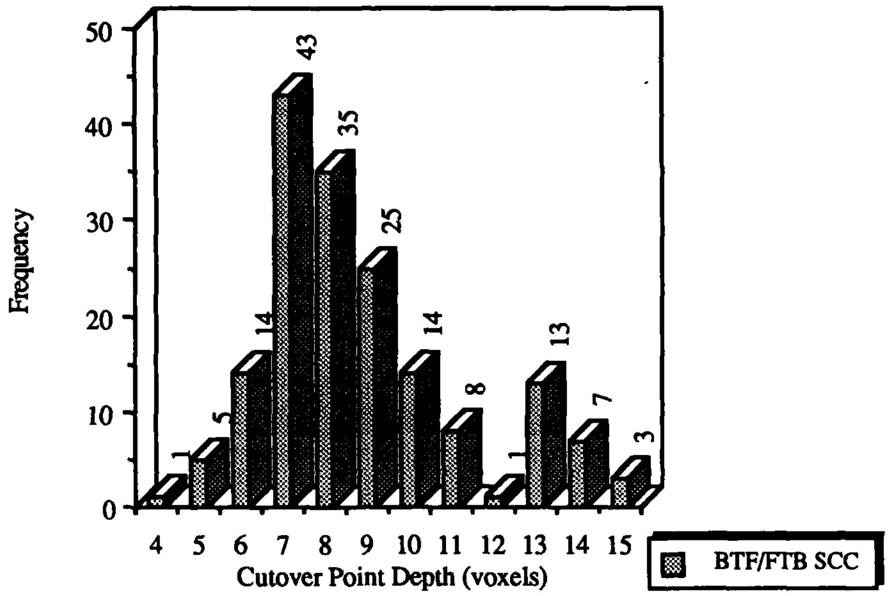
**BTF/FTB Cutover Depth vs Frequency
On a 90% Full 16 x 16 x 16 Cube**



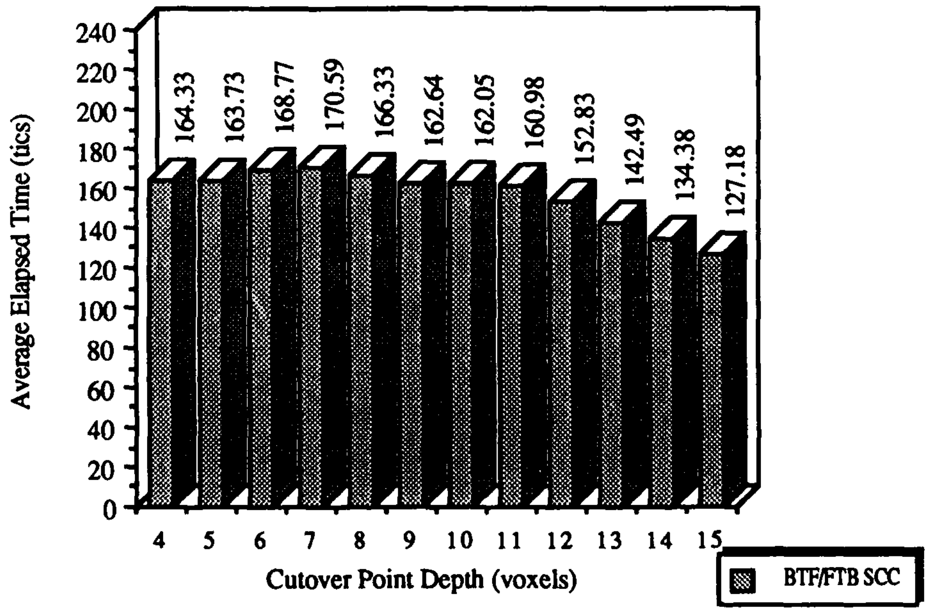
**BTF/FTB Cutover Depth vs Elapsed Time
On a 90% Full 16 x 16 x 16 Cube**



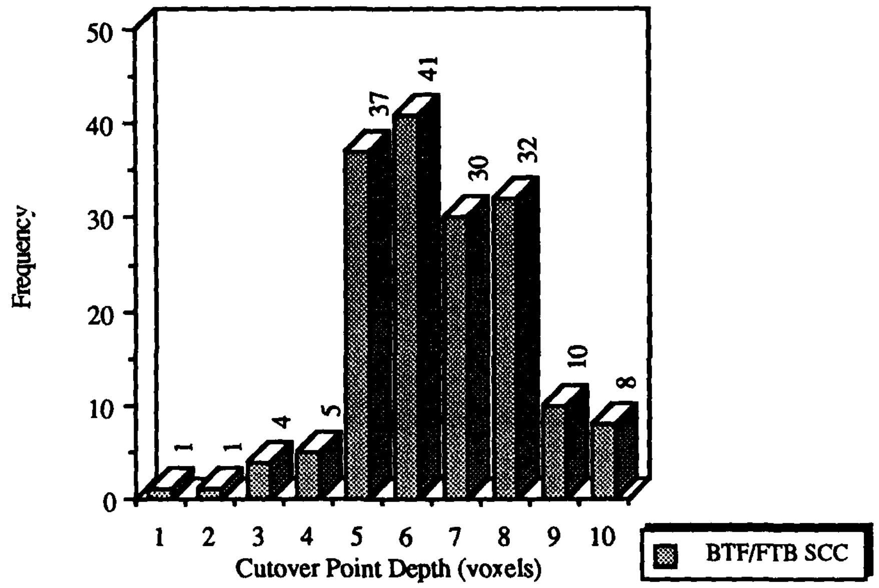
BTF/FTB Cutover Depth vs Frequency On a 50% Full 16 x 16 x 16 Cube



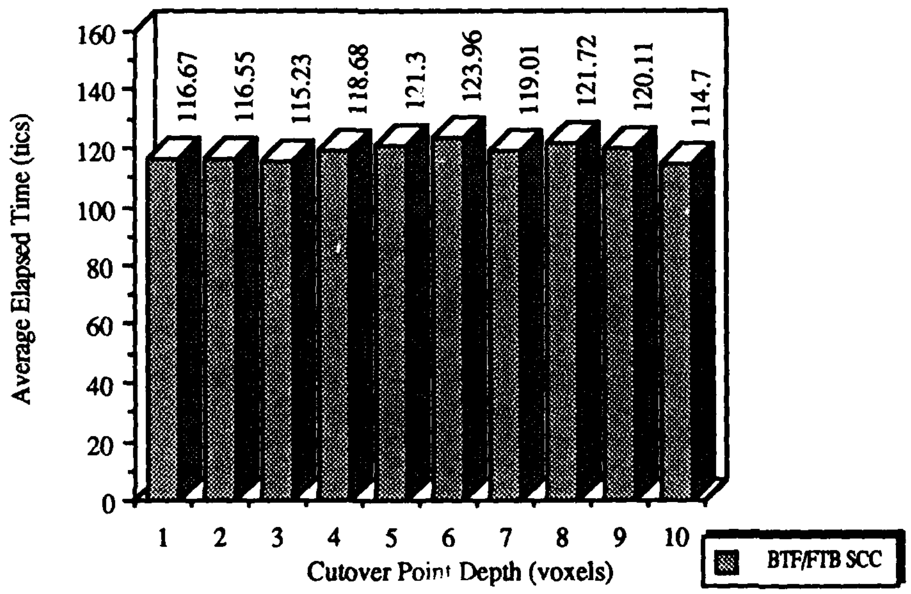
BTF/FTB Cutover Depth vs Elapsed Time On a 50% Full 16 x 16 x 16 Cube



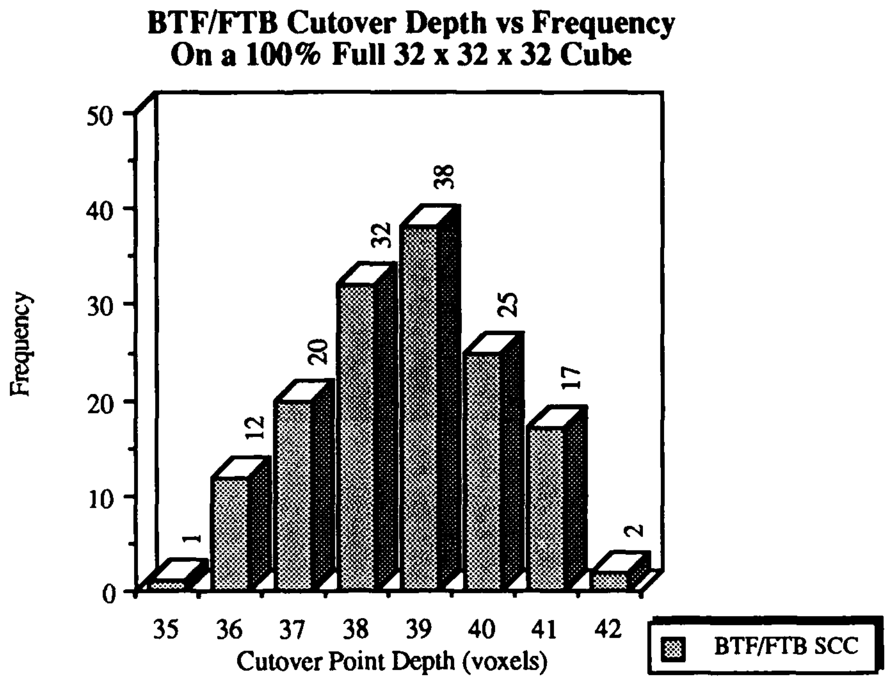
BTF/FTB Cutover Depth vs Frequency
On a 10% Full 16 x 16 x 16 Cube

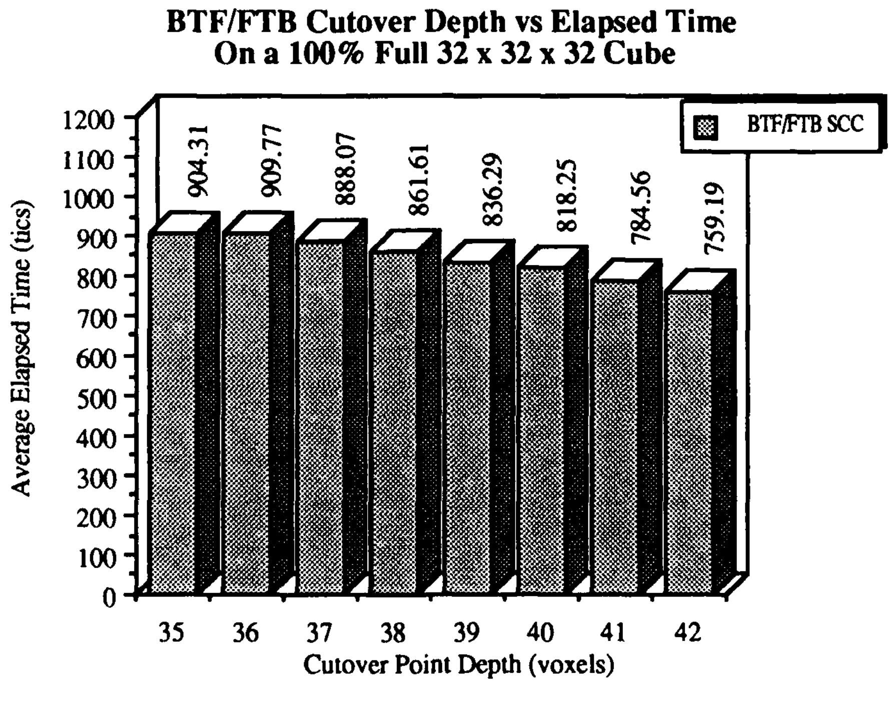


BTF/FTB Cutover Depth vs Elapsed Time
On a 10% Full 16 x 16 x 16 Cube

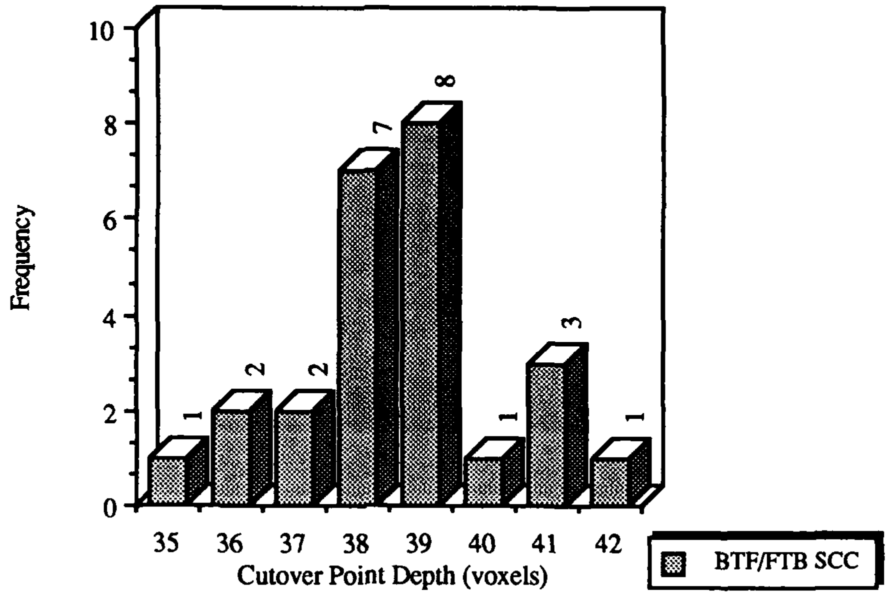


L.4. DYNAMIC ADAPTIVE RECURSIVE HSR ALGORITHM PERFORMANCE ON A 32 X 32 X 32 CUBE.

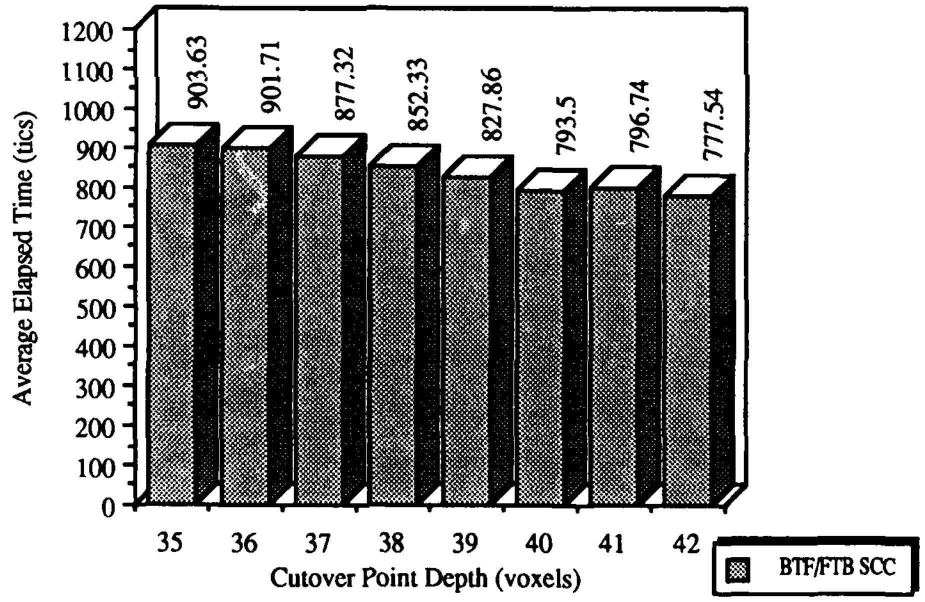




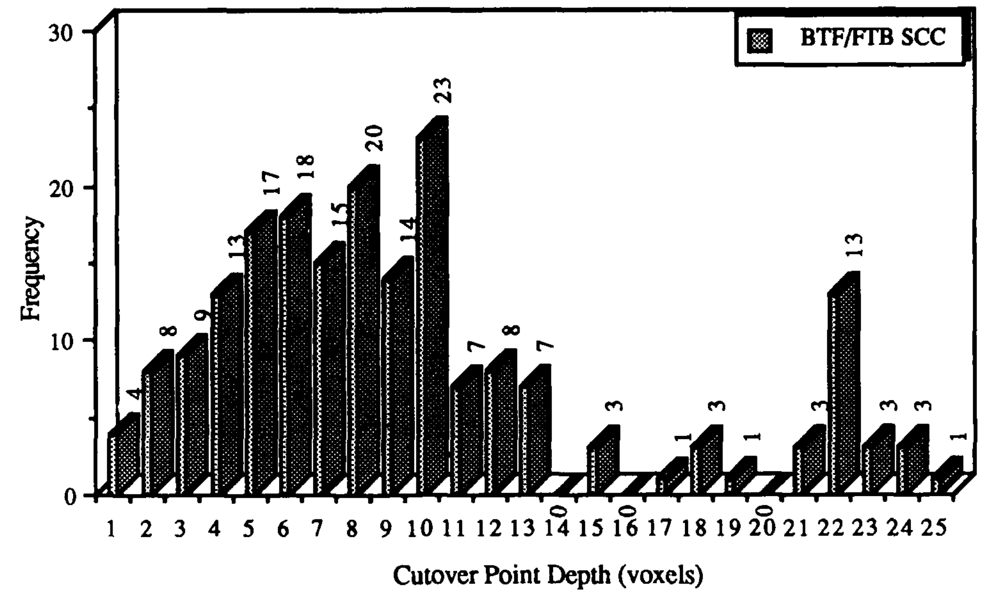
BTF/FTB Cutover Depth vs Frequency On a 90% Full 32 x 32 x 32 Cube



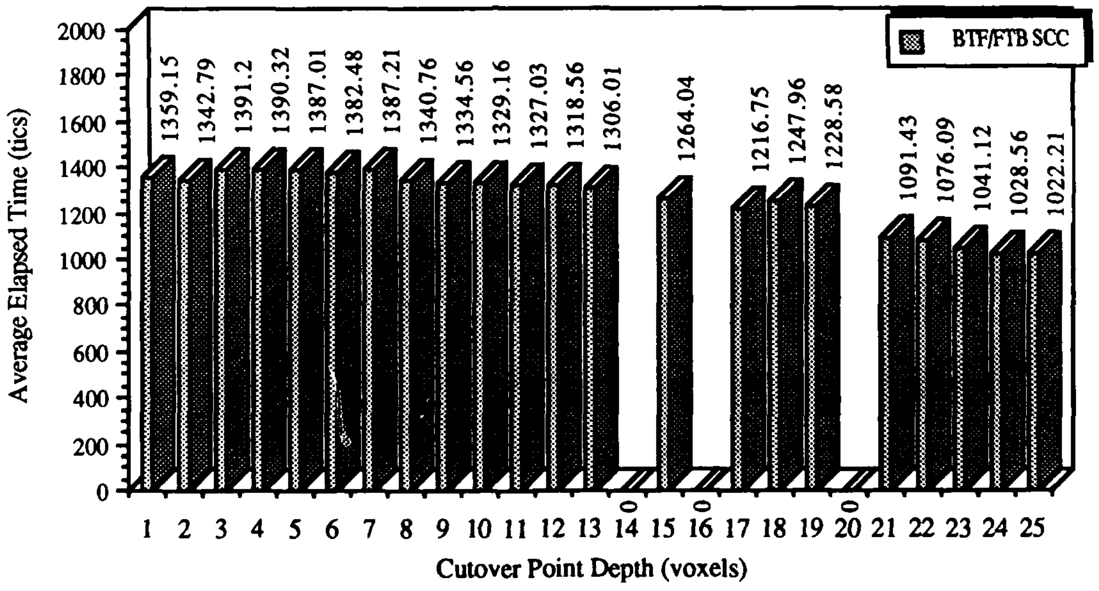
BTF/FTB Cutover Depth vs Elapsed Time On a 90% Full 32 x 32 x 32 Cube



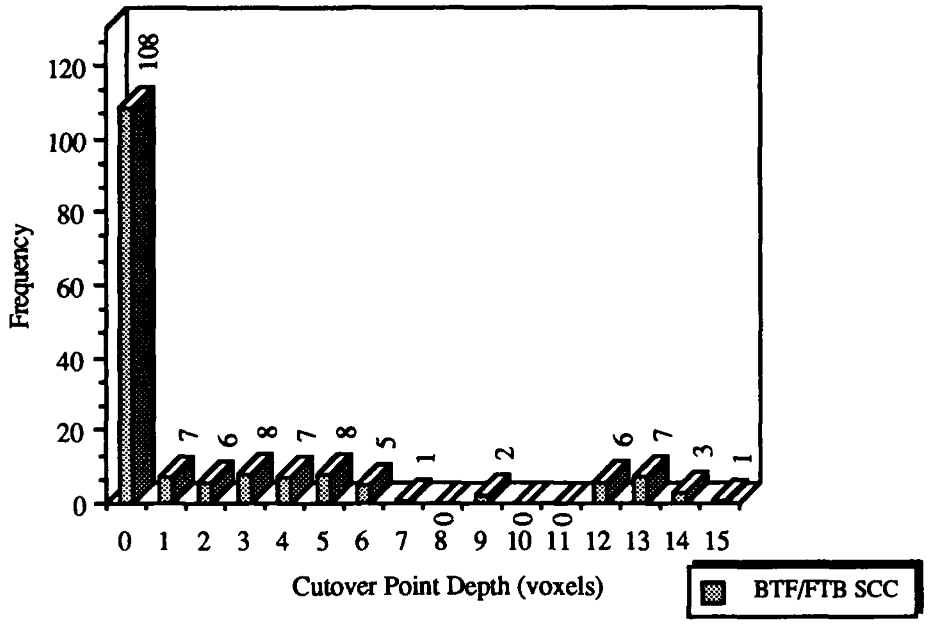
**BTF/FTB Cutover Depth vs Frequency
On a 50% Full 32 x 32 x 32 Cube**



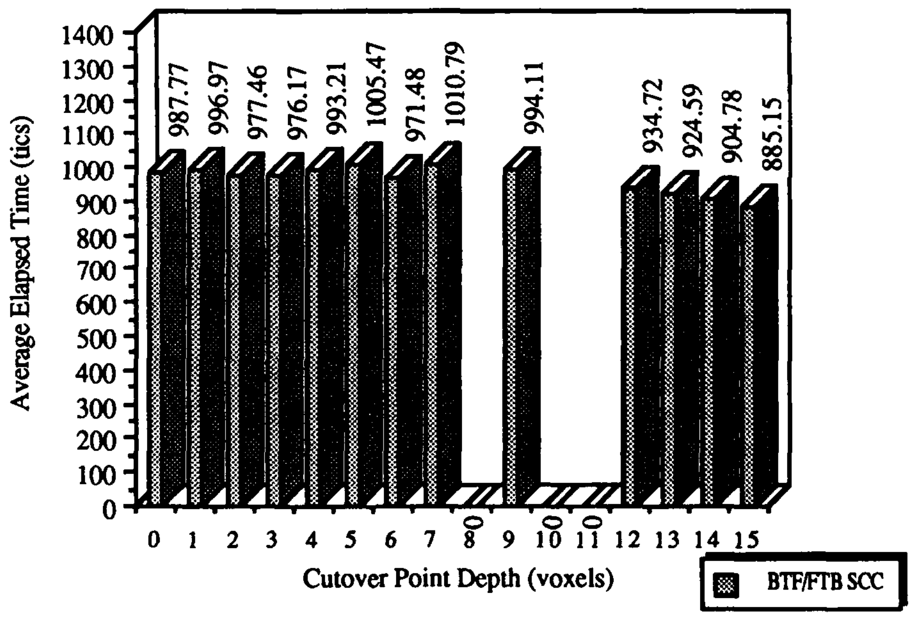
**BTF/FTB Cutover Depth vs Elapsed Time
On a 50% Full 32 x 32 x 32 Cube**



**BTF/FTB Cutover Depth vs Frequency
On a 10% Full 32 x 32 x 32 Cube**



**BTF/FTB Cutover Depth vs Elapsed Time
On a 10% Full 32 x 32 x 32 Cube**



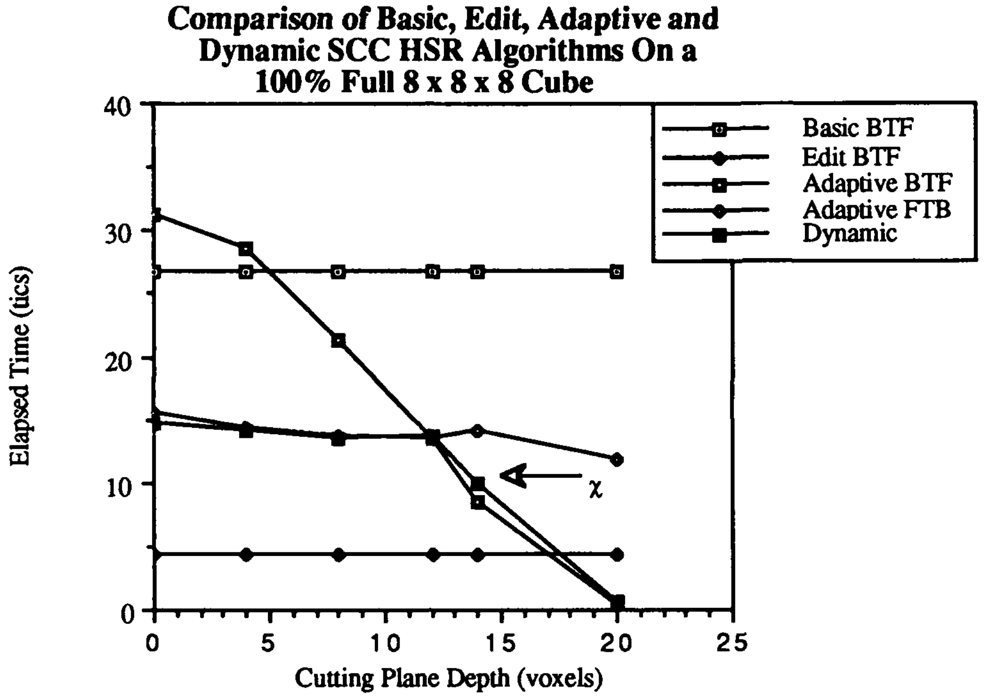
APPENDIX M

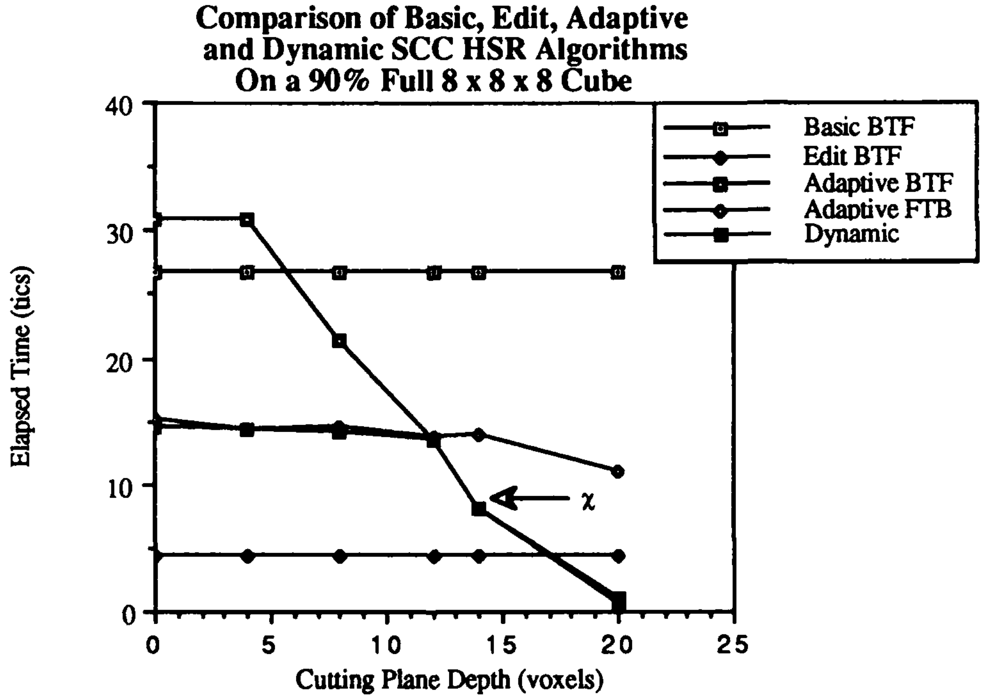
APPENDIX M
ILLUSTRATIVE PERFORMANCE OF THE DYNAMIC ADAPTIVE RECURSIVE HIDDEN-SURFACE REMOVAL ALGORITHM

M.1 INTRODUCTION.

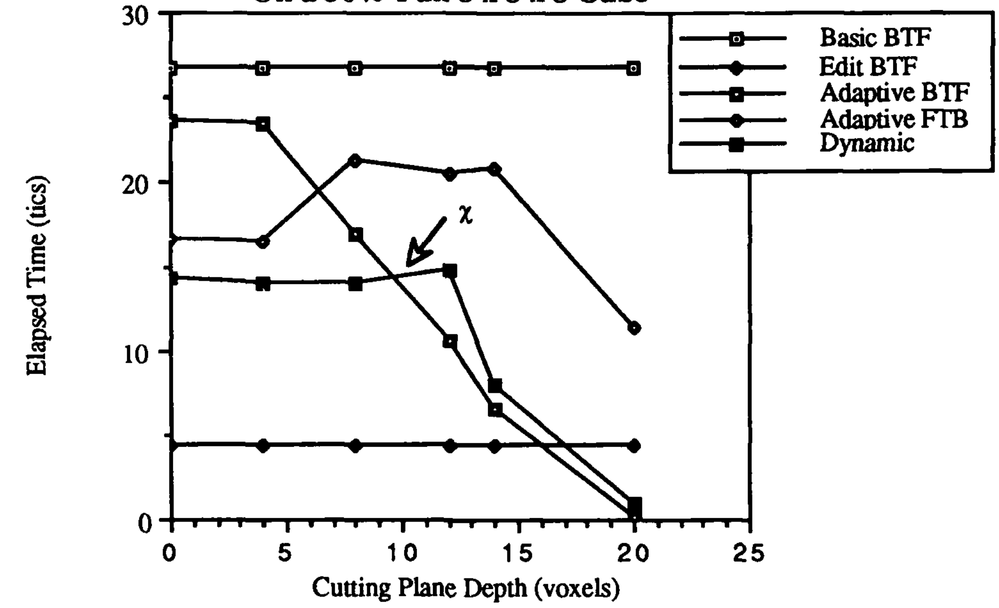
The performance results presented in this appendix illustrate the performance of the dynamic adaptive recursive SCC-based hidden-surface removal (HSR) algorithm when both algorithms are combined into one code module over a range of scene sizes and threshold window saturation values. Note expecially how the algorithm switches between the BTF and FTB components in response to cutting plane depth and σ value. The presentation is organized based upon the object space volume and threshold window saturation value. For each $\{\mathcal{N}_o\}$ size (8^3, 16^3, and 32^3) σ was set to 10%, 50%, 90%, and 100% and the resulting elapsed time to form the image was determined.

M.2. DYNAMIC ADAPTIVE RECURSIVE HSR ALGORITHM PERFORMANCE ON AN 8 X 8 X 8 CUBE.

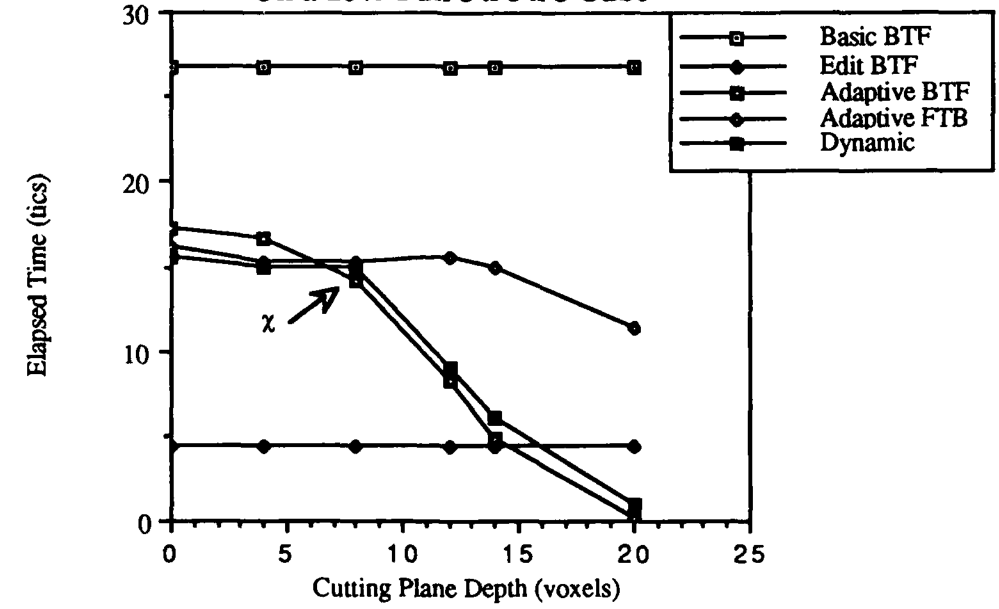




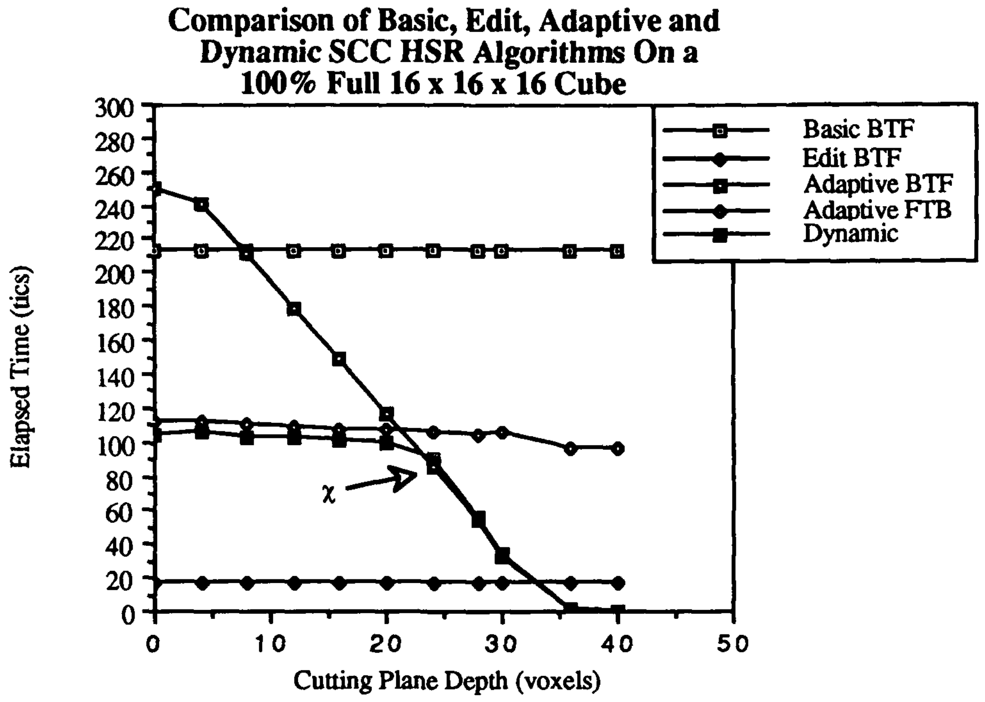
Comparison of Basic, Edit, Adaptive and Dynamic SCC HSR Algorithms On a 50% Full 8 x 8 x 8 Cube

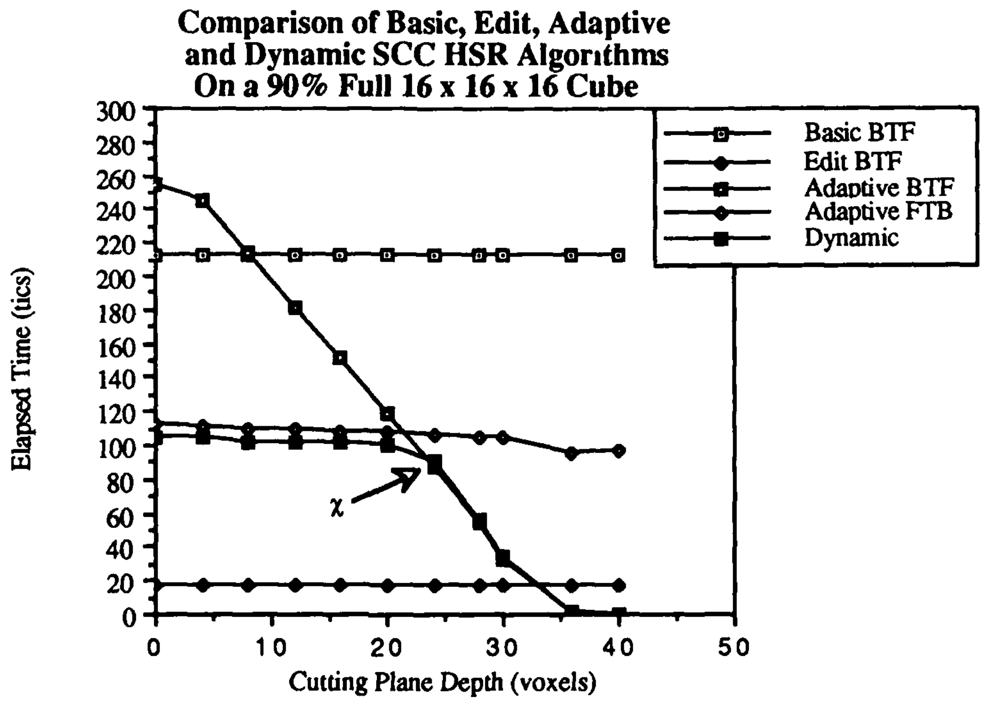


Comparison of Basic, Edit, Adaptive and Dynamic SCC HSR Algorithms On a 10% Full 8 x 8 x 8 Cube

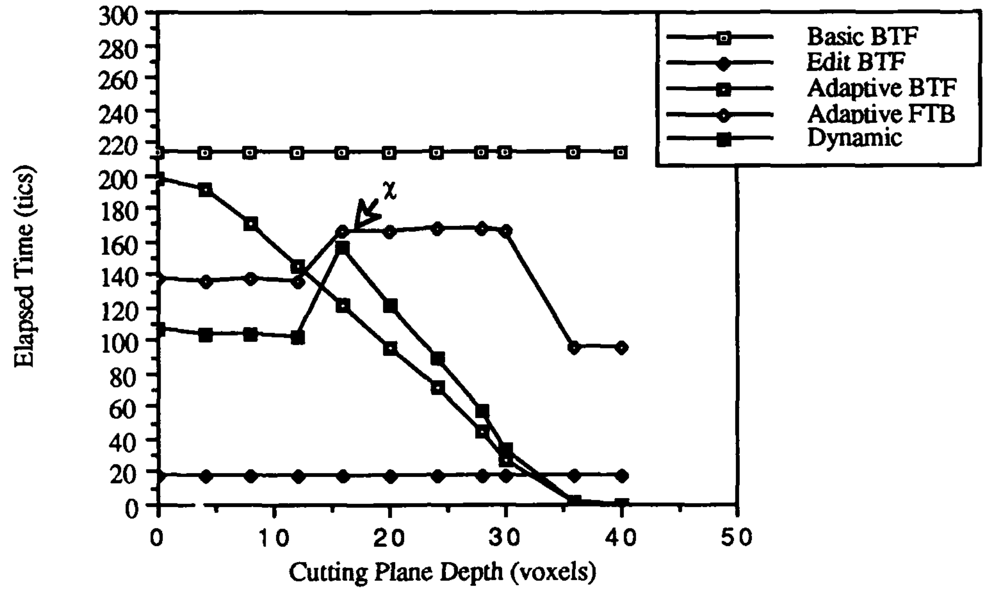


M.3. DYNAMIC ADAPTIVE RECURSIVE HSR ALGORITHM PERFORMANCE ON A 16 X 16 X 16 CUBE.

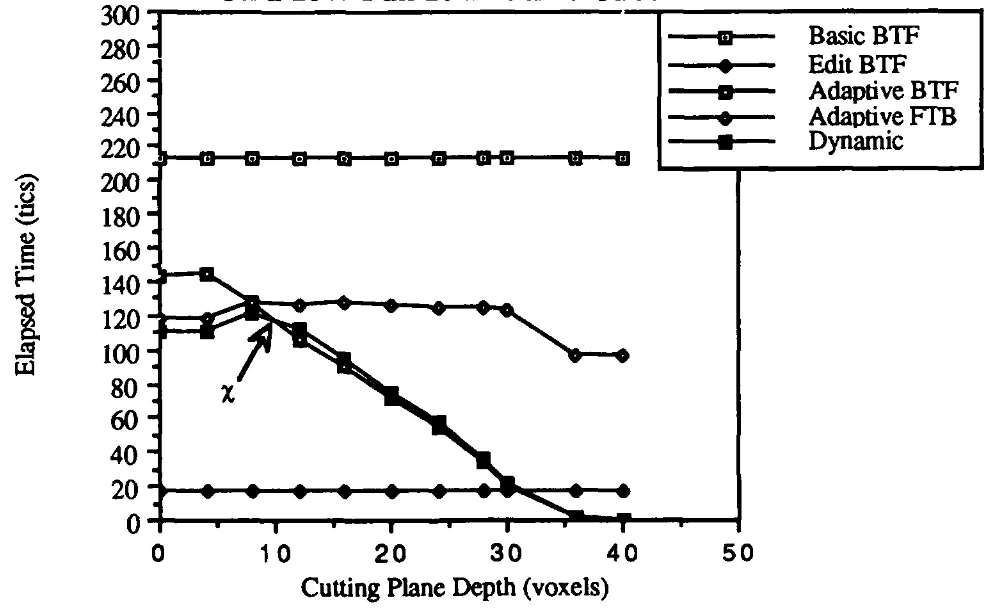




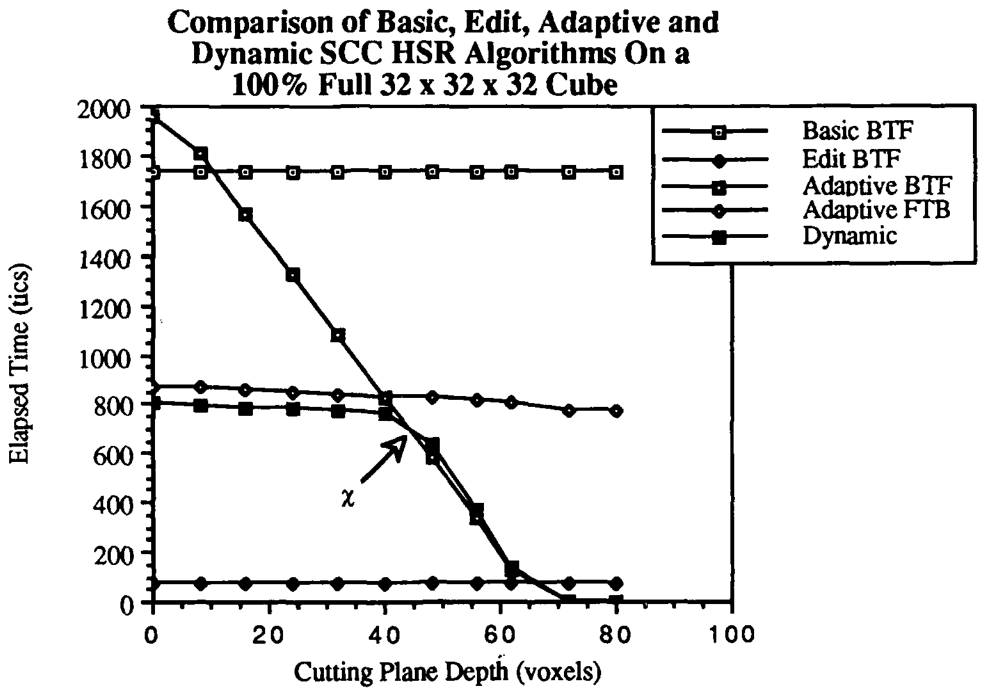
Comparison of Basic, Edit, Adaptive and Dynamic SCC HSR Algorithms On a 50% Full 16 x 16 x 16 Cube

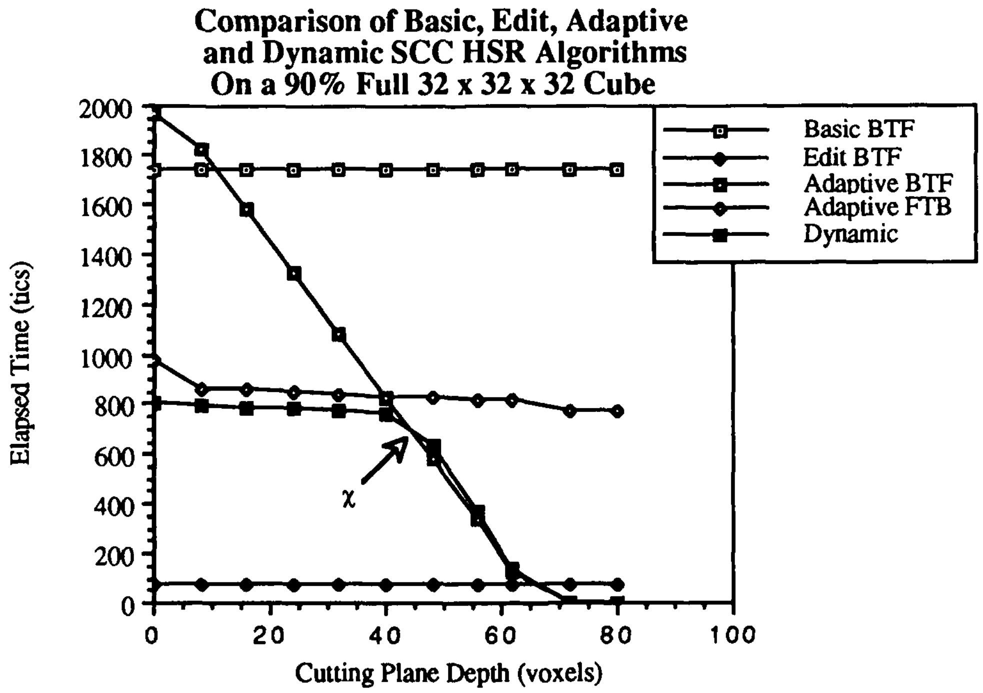


Comparison of Basic, Edit, Adaptive and Dynamic SCC HSR Algorithms On a 10% Full 16 x 16 x 16 Cube

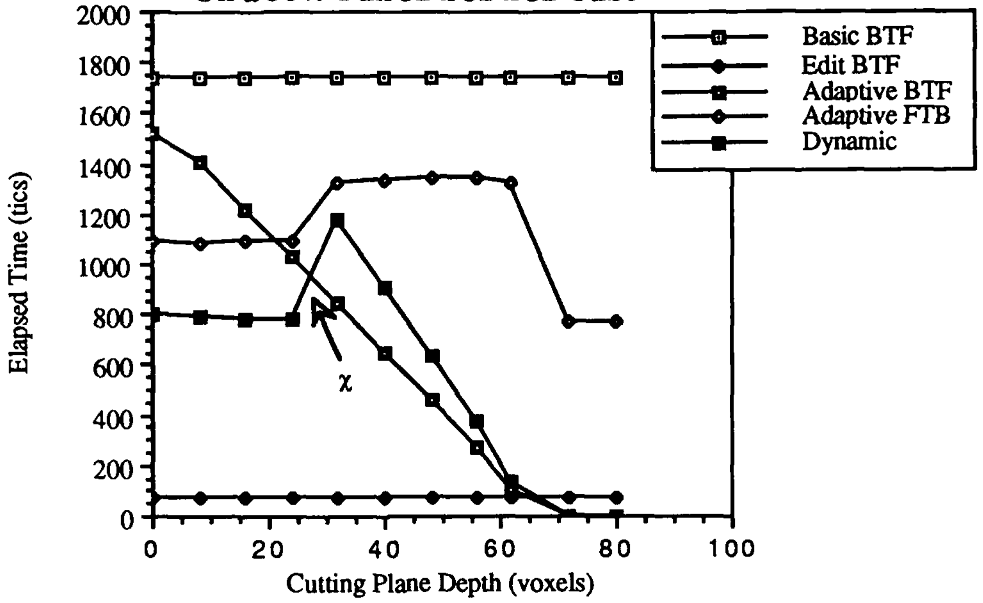


M.4. DYNAMIC ADAPTIVE RECURSIVE HSR ALGORITHM PERFORMANCE ON A 32 X 32 X 32 CUBE.

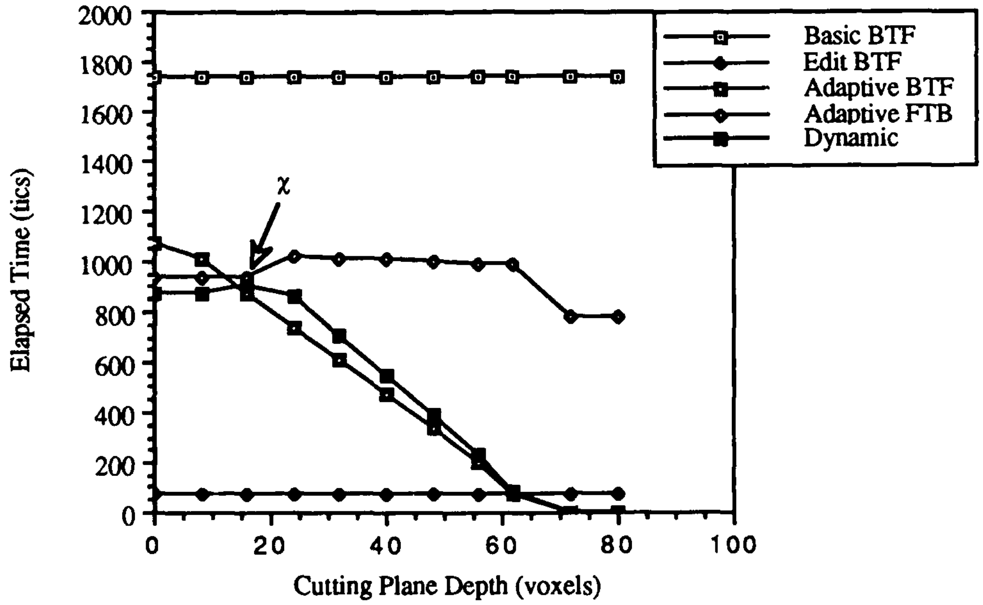




Comparison of Basic, Edit, Adaptive and Dynamic SCC HSR Algorithms On a 50% Full 32 x 32 x 32 Cube



Comparison of Basic, Edit, Adaptive and Dynamic SCC HSR Algorithms On a 10% Full 32 x 32 x 32 Cube



BIBLIOGRAPHY

BIBLIOGRAPHY

[Abd83] Abdou, Ikram E. "A Pipeline Machine for Image Processing Applications," *Proceedings of the International Conference on Parallel Processing*, Bellaire, Michigan, pp. 255-257, August 1983.

[Afr85] Afrati, Foto; Papadimitrou, Christos H.; and Papageorgiou,George. "The Complexity of Cubical Graphs," *Information and Control*, vol. 66, pp. 53-60, 1985.

[Agr81] Agrawal, Dharma P. and Jain, Ramesh. "A Multiprocessor System for Dynamic Scene Analysis," *1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Hot Springs, Virginia, pp. 96-103, November 1981.

[Ale88] Alexander, J. and Krumme, H.J. "Sonatom Plus: New Perspectives in Computerized Tomography," *Electromedica*, vol. 56, no. 2, pp. 50-56, 1988.

[All88] Allen, Randy; Borden, Bruce; Johnson, Steve; Kaplan, Michael; Ting, Way; and Wetherell, Charles. "System Software for the Sciences: Taming the Single-user Supercomputer," *Thirty-Third IEEE Computer Society International Conference*, San Francisco, California, pp. 457-461, February 29 - March 4, 1988.

[All87] Allo, Geza and Staszny, Gabor. "Two Algorithms for Image Segmentation," *Microprocessing and Microprogramming*, vol. 19, pp. 19-25, 1987.

[Ama85] Amans, Jean-Louis and Darrier, Pierre . "Processing and display of medical three dimensional arrays of numerical data using octree encoding," *Proceedings of the Society of Photo-Optical Instrumentation Engineers*, vol. 593, pp. 78-88, 1985.

[Apg88] Apgar, Brian; Bersack, Bret; and Mammen, Abraham. "A Display System for the Stellar™ Graphics Supercomputer Model GS1000™," *Computer Graphics*, vol. 22, no. 4, pp. 255-262, August 1988.

[Ard88a] Ardent™ Computer. *Titan™ Specifications.* 1988.

[Ard88b] Ardent™ Computer. *The Titan™ Graphics Supercomputer: A System Overview.* 1988.

[Ard88c] Ardent™ Computer. *DORE' Technical Overview.* April 1988.

[Art79a] Artzy, Ehud. "Display of Three-Dimensional Information in Computed Tomography," *Computer Graphics and Image Processing*,vol. 9, pp. 196-198, 1979.

[Art79b] Artzy, E.; Frieder, G.; Herman G.T.; and Liu, H.K. "A System for Three-Dimensional Dynamic Display of Organs From Computed Tomograms," *Proceedings of the Sixth Conference on Computer Applications in Radiology & Computer/Aided Analysis of Radiological Images*, Newport Beach, California, pp. 285-290, June 1979.

[Art81] Artzy, Ehud; Frieder, Gideon; and Herman, Gabor T. "The Theory, Design, Implementation and Evaluation of a Three-Dimensional Surface Detection Algorithm," *Computer Graphics and Image Processing*, vol. 15, pp. 1-24, January 1981.

[Arv87] Arvo, James and Kirk, David. "Fast Ray Tracing by Ray Classification," *Computer Graphics*, vol. 21, no. 4, pp. 55-64, July 1987.

[Att88a] AT&T Pixel Machines Inc. *AT&T Pixel Machines PXM 900™ Series Standard Software*. June 1988.

[Att88b] AT&T Pixel Machines Inc. *PIClib™ - The Pixel Machine's Graphics Library*. June 1988.

[Att88c] AT&T Pixel Machines Inc. *The Pixel Machine 900 Series*. June 1988.

[Att88d] AT&T Pixel Machines Inc. *Pixel Machine 900 Series Configurations*. June 1988.

[Att88e] AT&T Pixel Machines Inc. *Pixel Machine 900 Series Specifications*. June 1988.

[Att88f] AT&T Pixel Machines Inc. *RAYlib™- The Pixel Machine 900 Series Ray Tracing Library*. June 1988.

[Att89] AT&T Pixel Machines Inc. *The Pixel Machine System Architecture: A Technical Report*. 1989.

[Aus88] Austin, John D. and Van Hook, Tim. "Medical image processing on an enhanced workstation," *Medical Imaging II: Image Data Management and Display*, Society of Photo-Optical Instrumentation Engineers vol. 914, pp. 1317-1324, 1988.

[Axe83] Axel, Leon; Herman, Gabor T.; Udupa, Jayaram K.; Bottomley, Paul A.; and Edelstein, William A. "Three-Dimensional Display of Nuclear Magnetic Resonance(NMR) Cardiovascular Images,"*Journal of Computer Assisted Tomography*, vol. 7, no. 1, pp. 172-174, February 1983.

[Axe87] Axelson, B.; Israelson, A.; and Larsson, S.A. "Studies of a Technique for Attenuation Correction in Single Photon Emission Computed Tomography," *Physics in Medicine and Biology*, vol. 32, no. 6, pp. 737-749, 1987.

[Bai87] Bailey, Dale L; Hutton, Brian F.; and Walker, Paul J. "Improved SPECT Using Simultaneous Emission and Transmission Tomography," *Journal of Nuclear Medicine*, vol. 28, no. 5, pp. 844 - 851, May 1987.

[Bak89] Bakalash, Reuven and Kaufman, Arie. “Medicube: A 3D Medical Imaging Architecture,” *Computers and Graphics*, vol. 13, no. 2, 1989.

[Bar88] Barfuss, Helmut; Fisher, Hubertus; Hentschel, Dietmar; Ladebeck, Ralf; and Vetter, Jurgen. "Whole-Body MR Imaging and Spectroscopy with a 4-T System," *Radiology*, vol. 169, no. 3, pp. 811-816, December 1988.

[Bar81] Barrett, Harrison H., and Swindell, William. *Radiological Imaging*. 2 vols. New York, New York: Academic Press, 1981.

[Bat83] Bates, R.H.T; Garden, Kathryn L.; and Peters, Terence M. "Overview of Computerized Tomography wit' :mphasis on Future Developments," *Proceedings of the IEEE*, vol. 71, no. 3, pp. 356-372, March 1983.

[Beg86] Beguelin, Adam and Vasicek, D.J. "Communication between Nodes of a Hypercube," *Hypercube Multiprocessors 1987: Proceedings of the Second Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 162-168, September 29 - October 1, 1986.

[Bel88] Bell, C. Gordon; Miranker, Glen S.; and Rubinstein, Jonathan J. "Supercomputing for One," *Spectrum*, vol. 25, no. 4, pp. 46-50, April 1988.

[Ben82] Ben-Ari, M. *Principles of Concurrent Programming*. Englewood Cliffs, New Jersey: Prentice Hall International,1982.

[Ber86] Bergman, Larry; Fuchs, Henry; and Grant, Eric. “Image Rendering by Adaptive Refinement,” *Computer Graphics*, vol. 20, no. 4, pp. 29-37, August 1986.

[Bis86] Bishop, Gary and Weimer, David M. “Fast Phong Shading,” *Computer Graphics*, vol. 20, no. 4, pp. 103-106, August 1986.

[Blo83a] Bloch, Peter, and Udupa, Jayaram K. "Application of Computerized Tomography to Radiation Therapy and Surgical Planning," *Proceedings of the IEEE*, vol. 71, no. 3, pp. 351-355, March 1983.

[Boo86] Booth, Kellogg S.; Forsey, David R.; and Paeth, Alan W. "Hardware Assistance for Z-Buffer Visible -Surface Algorithms," *IEEE Computer Graphics and Applications*, vol. 6, no. 11, pp. 31-39, November 1986.

[Bor88a] Borden, Bruce S. "DORE' Platform Requirements," *Ardent™ Computer Corpor ›n White Paper*, 1988.

[Bor88b] Borden, Bruce S. "Portable DORE'" *Ardent™ Computer Corporation White Paper*, 1988.

[Bor88c] Borden, Bruce S. "Why DORE'? (or How DORE' and PHIGS relate)," *Ardent™ Computer Corporation White Paper*, December 1988.

[Blo83b] Bloch, Peter, and Udupa, Jayaram K. "Display of Three-Dimensional Data: A Survey of Methodologies and Applications," *Technical Report MIPG75, Dept. of Radiology, University of Pennsylvania*, January 1983.

[Bro84] Brotman, Lynne S. and Badler, Norman I. "Generating Soft Shadows with a Depth Buffer Algorithm," *IEEE Computer Graphics and Applications*, vol. 4, no. 10, pp. 5-12, October 1984.

[Bro53] Brownell, G.L and Sweet, W.H. "Localization of Brain Tumors with Positron Emitters," *Nucleonics*, vol. 11, no. 11, November 1953, pp. 40-45.

[Bro73] Brownell, G.L. and Burnham, C.A. "MGH Positron Camera," in *Tomographic Imaging in Nuclear Medicine*, G.S. Freeman, ed. New York, New York: Society of Nuclear Medicine, 1973, pp. 154-164.

[Bud80] Budinger, Thomas F. "Physical Attributes of Single-Photon Tomography," *The Journal of Nuclear Medicine*, vol. 21, no. 6, pp. 579-592, June 1980.

[Bus88] Bushong, Stewart C. *Magnetic Resonance Imaging: Physical and Biological Principles,* St. Louis, Missouri: The C.V. Mosby Co., 1988.

[Byd88] Bydder, G.M. "Magnetic Resonance Imaging: Present Status and Future Perspectives," *The British Journal of Radiology*, vol. 61, no. 730, pp. 889-897, October 1988.

[Cah86] Cahn, Deborah U. "PHIGS: The Programmer's Hierarchical Interactive Graphics System," *Proceedings of the Seventh Annual Conference and Exposition of the National Computer Graphics Association, NCGA '86*, Anaheim, California, pp. 333-355, May 11-15, 1986.

[Car85] Carlbom, Ingrid; Chakravarty, Indranil; and Vanderschel,David. "A Hierarchical Data Structure for Representing the Spatial Decomposition of 3-D Objects," *IEEE Computer Graphics and Applications*, vol. 5, no. 4, pp. 24-31, April 1985.

[Car84] Carpenter, Loren. "The A-buffer, an Antialiased Hidden Surface Method," *Computer Graphics*, vol. 18, no. 3, pp. 103-108, July 1984.

[Cat80] Catmull, Ed and Smith, Alvy Ray. "3-D Transformations of Images in Scanline Order," *Computer Graphics*, vol. 14, no. 3, pp. 279- 285, September 1980.

[Cat84] Catmull, Edwin. "An Analytic Visible Surface Algorithm for Independent Pixel Processing," *Computer Graphics*, vol. 18, no. 3, pp. 109 - 115, July 1984.

[Cha87] Chandra, Ramesh. *Introductory Physics of Nuclear Medicine*. Philadelphia, Pennsylvania: Lea & Febiger, 1987.

[Che84a] Chen, L.S.; Herman, G.T.; Reynolds, R.A.; and Udupa,J.K. "Surface Rendering in the Cuberille Environment," *Technical Report MIPG87*, Dept. of Radiology, University of Pennsylvania, January 1984.

[Che84b] Chen, L.S.; Herman, G.T.; Meyer, C.R.; Reynolds,R.A.; and Udupa, J.K. "3D83 - An Easy-To-Use Software Package for Three- Dimensional Display From Computed Tomograms," *IEEE Computer Society Joint International Symposium on Medical Images and Icons*, Arlington, Virginia, pp. 309-316, July 1984.

[Che85] Chen, Lih-Shyang; Herman, Gabor T.; Reynolds, R.Anthony; and Udupa, Jayaram K. "Surface Shading in the Cuberille Environment," *IEEE Computer Graphics and Applications*, vol.5, no. 12, pp. 33-43, December 1985.

[Che87] Chen, Ming-Syan and Shin, Kang G. "Embedding of Interacting Task Modules into a Hypercube," *Hypercube Multiprocessors 1987: Proceedings of the Second Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 122-129, September 29 - October 1, 1986.

[Coh90] Cohen, Daniel; Kaufman, Arie; and Bakalash, Reuven. "Real-Time Discrete Shading," to appear in *The Visual Computer*, vol. 6, no. 3, January 1990.

[Coo82] Cook, Robert L. and Torrance, K.E. "A Reflectance Model for Computer Graphics," *ACM Transactions on Graphics*, vol. 1, no. 1, pp. 7-24, January 1982.

[Coo84a] Cook, Robert L. "Shade Trees," *Computer Graphics*, vol. 18, no. 3, pp. 223-231, July 1984.

[Coo84b] Cook, Robert L.; Porter, Thomas, Carpenter, Loren. "Distributed Ray Tracing," *Computer Graphics*, vol. 18, no. 3, pp. 137-145, July 1984.

[Coo86] Cook, Robert L. "Stochastic Sampling in Computer Graphics," *ACM Transactions on Graphics*, vol. 5, no. 1, pp. 51-72, January 1986.

[Coo87] Cook, Robert L.; Carpenter, Loren; and Catmull, Edwin. "The Reyes Image Rendering Architecture," *Computer Graphics*, vol. 21, no. 4, pp. 95-102, July 1987.

[Coo88] Cooprider, L.; Gurwitz, R.; and Teixeira, T. "Support for Tightly Coupled Processors," *Unix Review*, vol. 6, no. 5, pp. 71-75, May 1988.

[Cro86] Croft, Barbara Y. *Single-Photon Emission Computed Tomography*, Chicago, Illinois: Year Book Medical Publishers, Inc., 1986.

[Cro77a] Crow, Franklin C. "Shadow Algorithms for Computer Graphics," *Computer Graphics*, vol. 11, no. 2, pp 242-247, 1977.

[Cro77b] Crow, Franklin C. "The Aliasing Problem in Computer-Generated Shaded Images," *Communications of the ACM*, vol. 20, no. 11, pp. 799-805, November 1977.

[Cro81] Crow, Franklin C. "A Comparison of Antialiasing Techniques," *IEEE Computer Graphics and Applications*, vol. 1, no. 1, pp. 40-48, January 1981.

[Dea83] Dean, S.R. *The Radon Transform and Some oj ..s Applications*, New York, New York: John Wiley & Sons, 1983.

[Dav87] Davis, Dwight B. "Parallel Computers Diverge," *High Technology,* pp. 16-22, February 1987.

[Del84] DeLand, Frank H. and Shih, Wei-Jen. "The Status of SPECT in Tumor Diagnosis," *The Journal of Nuclear Medicine*, vol. 25, no. 12, pp. 1375 - 1379, 1984.

[DelG87] Del Guerra, A. "Positron Emission Tomography," *Physica Scripta*, vol. T19, pp. 481-486, 1987.

[DeP77] DePresseux, Jean-Claude. "The Positron Emission Tomography and its Applications," *Journal of Belge Radiology*, vol. 60, pp. 483-500, 1977.

[Der87] Derenzo, Stephen E. "Recent Developments in Positron Emission Tomography (PET) Instrumentation," *Physics and Engineering of Computerized Multidimensional Imaging and Processing*, SPIE vol. 671, pp. 232-242, 1986.

[Die88] Diede, Tom; Hagenmaier, Carl F.; Miranker, Glen S.; Rubinstein, Jonathan J.; and Worley, William S. Jr. "The Titan Graphics Supercomputer Architecture," *Computer*, vol. 21, no. 9, pp. 13-29, September 1988.

[Dip84] Dippe', Mark and Swensen, John. "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis," *Computer Graphics*, vol. 18, no. 3, pp. 310-319, July 1984.

[Doc81] Doctor, Louis J., and Torborg, John G. "Display Techniques for Octree-Encoded Objects," *IEEE Computer Graphics and Applications*, vol. 1, no. 3, pp. 29-38, 1981.

[Doh86] Doherty, Mark F; Bjorklund, Carolyn M.; and Noga, Mark T. "Split-Merge-Merge: An Enhanced Segmentation Capability," *Proceedings of the 1986 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Miami Beach, Florida, pp. 325-328, June 22-26, 1986

[Dre88] Drebin, Robert A.; Carpenter, Loren; and Hanrahan, Pat. "Volume Rendering," *Computer Graphics*, vol. 22, no. 4, pp. 65-74, August 1988.

[Duf85] Duff, Tom. "Compositing 3-D Rendered Images," *Computer Graphics*, vol. 19, no. 4, pp. 41-44, July 1985.

[Dun86] Dunigan, T.H. "Hypercube Performance," *Hypercube Multiprocessors 1987: Proceedings of the Second Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 178-192, September 29 - October 1, 1986.

[Edh86] Edholm, P.R.; Lewitt, R.M.; Lindholm, B.; Herman,G.T.; Udupa, J.K.; Chen, L.S.; Margasahayam, P.S.; and Meyer,C.R. "Contributions to Reconstruction and Display Techniques in Computerized Tomography," *Technical Report MIPG110*, Dept. of Radiology, University of Pennsylvania, May 1986.

[Eli89] Eliasen, Rune. "Graphics Supercomputing for Mechanical Engineers: Ardent Computer Corp.," *Journal of Engineering Computing and Applications*, vol. 3, no. 3, pp. 21-27, Spring 1989.

[El182] Ell, P.J.; Khan, O.; Jarritt, P.H.; and Cullum, I.D. *Radionuclide Section Scanning: An Atlas of Clinical Practice*. London, England: Chapman and Hall, 1982.

[Els86] Elster, Allen D. *Magnetic Resonance Imaging: A Reference Guide and Atlas*. Philadelphia, Pennsylvania: J. B. Lippincott Company, 1986.

[Eng86] English, Robert J. and Brown, Susan E. *SPECT Single-Photon Emission Computed Tomography: A Primer.* New York, New York: The Society of Nuclear Medicine, 1986.

[Eri84] Erickson, Jon. "Imaging Systems," In Harbert, John and DaRocha, Antonio F. G., (Eds): *Textbook of Nuclear Medicine, Volume I: Basic Science*, 2nd edition, Philadelphia, Pennsylvania: Lea & Febiger, pp 105 - 159, 1984.

[Ess84] Esser, Peter D.; Alderson, Philip O.; Mitnick, Robin J.; and Arliss, Jeffrey J. "Angled-Colimator SPECT (A-SPECT): An Improved Approach to Cranial Single Photon Emission Tomography," *Journal of Nuclear Medicine*, vol. 25, no. 7, pp. 805-809, June, 1984.

[Eva85] Evans, Robley D. *The Atomic Nucleus*, Malabar, Florida: R.E. Krieger, 1985.

[Far84] Farre'', Edward J.; Zappulla, Rosario; and Yang,Wen C. "Color 3-d Imaging of Normal and Pathologic Intracranial Structures," *IEEE Computer Graphics and Applications*, vol. 4, no. 10, pp. 5-17, September 1984.

[Far85] Farrell, Edward J.; Yang, Wen C.; and Zappulla, Rosario. "Animated 3D CT Imaging," *IEEE Computer Graphics and Applications*, vol. 5, no. 12, pp. 26-30, December 1985.

[Far86a] Farrell, Edward J.; Zappulla, Rosario A.; Kantrowitz, Allen. "Planning Neurosurgery with Interactive 3D Computer Imaging," *Proceedings of the 5th Conference on Medical Informatics*, Salamon R.; and Blum, B. (Eds.), Amsterdam, Denmark: North-Holland, 1986.

[Far86b] Farrell, Edward J.; Zappulla, Rosario A.; Kantrowitz, Allen; Spigelman, Mel; Yang, Wen C. "3D Display of Multiple Intracranial Structures for Treatment Planning," *Proceedings of the Eighth Annual Conference of the IEEE/Engineering in Medicine and Biology Society*, Fort Worth, Texas, pp. 1088-1091, November 7-10, 1986.

[Far87] Farrell, Edward J.; Zappulla, Rosario A.; and Spigelman, M. "Imaging Tools for Interpreting Two- and Three-Dimensional Medical Data," *Proceedings of the Eighth Annual Conference and Exposition of the National Computer Graphics Association*, National Computer Graphics Association, McLean, Virginia, pp. 60-69, 1987.

[Far89] Farrell, Edward J. and Zappulla, Rosario A. "Three-Dimensional Data Visualization and Biomedical Applications," *CRC Critical Reviews in Biomedical Engineering*, vol. 16, no. 4, pp. 323-363, 1989.

[Fle84] Fleischer, Arthur C. and James, A. Everette Jr. *Real-Time Sonography* , Norwalk, Connecticut: Appleton-Century-Crofts, 1984.

[Fly72] Flynn, M.J. "Some Computer Organizations and Their Effectiveness," *IEEE Transactions on Computers*, C-21, no. 9, pp. 948-960, September 1972.

[Fly83] Flynn, M.; Matteson, R.; Dickie, D.; Keyes, J.W. Jr.; and Bookstein, F. "Requirements for the display and analysis of three-dimensional medical image data," *2nd International Conference and Workshop on Picture Archiving and Communication Systems (PACSII) for Medical Applications*, SPIE vol. 418, Kansas City, Missouri, pp. 213-224, May 22-25, 1983.

[Fol83] Foley, James D., and Van Dam, Andries. *Fundamentals of Interactive Computer Graphics*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1983.

[Fox87] Fox, Geoffrey C. "The Caltech Concurrent Computation Program," *Hypercube Multiprocessors 1987: Proceedings of the Second Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 353-381, September 29 - October 1, 1986.

[Fri85a] Frieder, G.; Gordon, D.; and Reynolds, R.A. "Back-to-Front Display of Voxel-Based Objects," *IEEE Computer Graphics and Applications*, vol. 5, no. 1, pp. 52-60, January 1985.

[Fri85b] Frieder, Gideon; Herman, Gabor T.; Meyer, Craig; and Udupa, Jayaram. "Large Software Problems for Small Computers: An Example from Medical Imaging," *IEEE Software*, vol. 2, no. 5, pp. 37-47, September 1985.

[Fri87] Frieder, Ophir. *Parallelization of [Uva87] segmentation algorithm.* Personal communication with author. June, 1987.

[Fri88a] Frieder, G; Frieder, O; Stytz, M.R. "A High Performance Parallel Approach to Medical Imaging," *Frontiers 88, IEEE Second Symposium on the Frontiers of Massively Parallel Computation*, Fairfax, Virginia, October, 1988.

[Fuc77] Fuchs, H.; Kedem, Z.M.; and Uselton, S.P. "Optimal Surface Reconstruction from Planar Contours," *Communications of the ACM*, vol. 20, no. 10, pp. 693-702, November 1977.

[Fuc79] Fuchs, Henry; Kedem, Zvi M.; and Naylor, Bruce. "Predetermining Visibility Priority in 3-D Scenes(Preliminary Report)," *Computer Graphics*, vol. 13, no. 2, pp. 175-181, August 1979.

[Fuc80] Fuchs, Henry; Kedem, Zvi M.; and Naylor, Bruce F. "On Visible Surface Generation by A Priori Tree Structures," *Computer Graphics*, vol. 14, no. 2, pp. 124-133, August 1980.

[Fuc83] Fuchs, Henry; Abram, Gregory D.; and Grant, Eric D. "Near Real-Time Shaded Display of Rigid Objects," *Computer Graphics*, vol. 17, no. 3, pp. 65-69, July 1983.

[Fuc85] Fuchs, Henry; Goldfeather, Jack; Hultquist, Jeff P.; Spach, Susan; Austin, John D.; Brooks, Frederick P. Jr.; Eyles, John G.; and Poulton, John. "Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes," *SIGGRAPH'85*, vol. 19, no. 3, pp. 111-120, July 22-26 1985.

[Fuc86] Fuchs, Henry; Poulton, John; Eyles, John; Austin,John; and Greer, Trey. "Pixel-Planes: A Parallel Architecture for Raster Graphics," *Pixel-Planes Project Summary*, August, 1986.

[Fuc88a] Fuchs, Henry; Poulton, John; Eyles, John; and Greer, Trey. "Coarse-Grain and Fine-Grain Parallelism in the Next Generation Pixel-Planes Graphics System," *TR88-014*, The University of North Carolina at Chapel Hill, Department of Computer Science, March 1988. To appear in: *Proceedings of the International Conference and Exhibition on Parallel Processing for Computer Vision and Display*, University of Leeds, United Kingdom, January 12-15, 1988.

[Fuc88b] Fuchs, Henry; Poulton, John; Eyles, John; Greer, Trey; Hill, Edward; Israel, Laura; Ellsworth, David; Good, Howard; Interrante, Victoria; Molnar, Steve; Neumann, Ulrich; Rhoades, John; Tebbs, Brice; and Turk, Greg. "Pixel-Planes: A Parallel Architecture for Raster Graphics," *Pixel-Planes Project Summary*, August, 1988.

[Fuc89a] Fuchs, Henry; Poulton, John; Eyles, John; Greer, Trey; Goldfeather, Jack; Ellsworth, David; Molnar, Steve; and Turk, Greg. "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories," *TR89-005*, The University of North Carolina at Chapel Hill, Department of Computer Science, May 1989. To appear in: *Computer Graphics*, vol. 23, no. 4.

[Fuc89b] Fuchs, Henry; Levoy, Marc; and Pizer, Stephen M. "Interactive Visualization of 3D Medical Data," *IEEE Computer*, vol. 22, no. 8, 46-51, August 1989.

[Fuj86] Fujimoto, Akira; Tanaka, Takayuki; and Iwata, Kansei. "ARTS: Accelerated Ray-Tracing System," *IEEE Computer Graphics and Applications*, vol. 6, no. 4, pp. 16-26, April 1986.

[Ful88] Fullerton, Gary D. and Potter, Janet L. "Computed Tomography," In Putman, Charles E. and Ravin, Carl E. (Eds): *Textbook of Diagnostic Imaging*, Philadelphia, Pennsylvania: W.B. Suanders Co, pp. 47-60,1988.

[Gar75] Garey, M.R., and Graham, R.L. "On Cubical Graphs," *Journal of Combinatorial Theory*, (B), vol. 18, pp. 84-95, 1975.

[GE88] General Electric Corporation, *MR Max Plus Magnetic Resonance System*, 1988.

[Gel89a] Gelberg, Lawrence M; MacMann, Jeffrey F.; and Mathias, Craig J. "Graphics, Image Processing, and the Stellar™ Graphics Supercomputer," *SPSE/SPIE Electronic Imaging Devices and Systems 1989 Symposium*, Anaheim, CA., January 1989, in press.

[Gel89b] Gelberg, Lawrence M.; Kamins, David; and Vroom, Jeff. "VEX: A Volume Exploratorium, An Integrated Toolkit for Interactive Volume Visualization," *Proceedings of the Chapel Hill Conference on Volume Visualization*, Chapel Hill, NC, pp. 21-26, May 18-19, 1989.

[Gha88] Gharachorloo, Nader; Gupta, Satish; Hokenek, Erdem; Balasubramanian, Peruvemba; Bogholtz, Bill; Mathieu, Christian; and Zoulas, Christos. "Subnanosecond Pixel Rendering with Million Transistor Chips," *Computer Graphics*, vol. 22, no. 4, pp. 41-49, August 1988.

[Gil76] Gilbert, Barry K.; Storma, Martin T.; James, CarlE.; Hobrock, Leon W.; Yang, Edward S.; Ballard, Keith C.; and Wood, Earl H. "A Real-Time Hardware System for Digital Processing of Wide-Band Video Images," *IEEE Transactions on Computers*, vol. C-25, no. 11, pp. 1089-1100, November 1976.

[Gio87] Giordano, Alberto; Giunchiglia, Fausto; Maresca,Massimo; and Vernazza, Tullio. "Evaluation of a Multiprocessor Structure for Image Processing," *Microprocessing and Microprogramming*, vol. 19, pp. 5-17, 1987.

[Gla84] Glassner, Andrew S. "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics and Applications*, vol. 4, no. 10, pp. 15-22, October 1984.

[Gla88] Glassner, Andrew S. "Spacetime Ray Tracing for Animation," *IEEE Computer Graphics and Applications*, vol. 8, no. 2, pp. 60-70, March 1988.

[Gol86] Goldfeather, Jack and Fuchs, Henry. "Quadratic Surface Rendering on a Logic-Enhanced Frame-Buffer Memory," *IEEE Computer Graphics and Applications*, vol. 6, no. 1, pp. 48-59, January 1986.

[Gol83] Goldwasser, Samuel M., and Reynolds, R. Anthony. "An Architecture for the Real-Time Display and Manipulation of Three-Dimensional Objects," *Proceedings of the International Conference on Parallel Processing*, Bellaire, Michigan, pp. 269-274, August 1983.

[Gol84a] Goldwasser, Samuel M. "A Generalized Object Display Processor Architecture," *Proceedings of the 11th Annual International Symposium on Computer Architecture*, Ann Arbor, Michigan, pp.38-47, May 1984.

[Gol84b] Goldwasser, Samuel M. "A Generalized Object Display Processor Architecture," *IEEE Computer Graphics and Applications*, vol. 4, no. 10, pp. 43-55, October 1984.

[Gol85a] Goldwasser, Sam M. "The Voxel Processor Architecture for Real-Time Display and Manipulation of 3D Medical Objects," *Proceedings of the Sixth Annual Conference and Exposition of the National Computer Graphics Association*, Dallas, Texas, pp. 71-80, April 14-18, 1985.

[Gol85b] Goldwasser, Samuel M.; Reynolds, R. Anthony; Bapty,Ted; Baraff, David; Summers, John; Talton, David, A.; and Walsh, Ed. "Physician's Workstation with Real-Time Performance," *IEEE Computer Graphics and Applications*, vol. 5, no. 12, pp. 44-57, December, 1985.

[Gol86a] Goldwasser, Samuel M. "Rapid Techniques for the Display and Manipulation of 3-D Biomedical Data," *Proceedings of the Seventh Annual Conference and Exposition of the National Computer Graphics Association, NCGA '86*, Anaheim, California, pp. 115-149, May 11-15 1986.

[Gol86b] Goldwasser, S.M. and Walsh, E.S. "High Speed Volume Rendering of 3-D Biomedical Data," *Proceedings of the Eighth Annual Conference of the IEEE/Engineering in Medicine and Biology Society*, Fort Worth, Texas, pp. 1084-1087, November 7-10, 1986.

[Gol87] Goldwasser, Samuel M., and Reynolds, R. Anthony. "Real-Time Display and Manipulation of 3-D Medical Objects: The Voxel Processor Architecture," *Computer Vision, Graphics and Image Processing*, vol. 39, pp. 1-27, 1987.

[Gol88a] Goldwasser, S. M.; Reynolds, R. A.; Talton, D. A.; and Walsh, E. S. "High- Performance Graphics Processors for Medical Imaging Applications," *Proceedings of the International Conference on Parallel Processing for Computer Vision and Display*, University of Leeds, United Kingdom, pp. 1-13, 12-15 January 1988.

[Gol88b] Goldwasser, Samuel M.; Reynolds, R. Anthony; Talton, David A.; and Walsh, Edward S. "Techniques for the Rapid Display and Manipulation of 3-D Biomedical Data," *Computerized Medical Imaging and Graphics*, vol. 12, no. 1, pp. 1-24, January-February 1988.

[Goo80] Goodwin, Paul N. "Recent Developments in Instrumentation for Emission Computed Tomography," *Seminars in Nuclear Medicine*, vol. 10, no. 4, pp. 322-334, October 1980.

[Gor83a] Gordon, Dan. "Scan Conversion Based on a Concept of Critical Points," *Technical Report No. CSD83-1*, Department of Computer Studies, University of Haifa, Haifa, Israel, May 1983.

[Gor83b] Gordon, Dan, and Reynolds, R. Anthony. "Image Space Shading of Three-Dimensional Objects," *Technical Report MIPG85*, Dept. of Radiology, University of Pennsylvania, November 1983.

[Gor85] Gordon, Dan, and Reynolds, R. Anthony. "Image Space Shading of 3-Dimensional Objects," *Computer Vision, Graphics, and Image Processing*, vol. 29, pp. 361-376, 1985.

[Gor89] Gordon, Dan and Udupa, Jayaram K. "Fast Surface Tracking in Three-Dimensional Binary Images," *Computer Vision, Graphics, and Image Processing*, vol. 45, pp. 196-214, 1989.

[Gor75] Gordon, Richard; Herman, Gabor T.; and Johnson, Steven A. "Image Reconstruction from Projections," *Scientific American*, pp. 56-68, October 1975.

[Gou71] Gouraud, Henri. "Continuous Shading of Curved Surfaces," *IEEE Transactions on Computers*, vol. C-20, no. 6, pp. 623-629, June 1971.

[Gra80] Granlund, Goesta H. "GOP, A Fast and Flexible Processor for Image Analysis," *Proceedings of the Fifth International Conference on Pattern Recognition*, Miami Beach, Florida, pp. 489-493, December 1980.

[Gre82] Greenleaf, James F. "Three-Dimensional Imaging in Ultrasound," *Journal of Medical Systems*, vol. 6, no. 6, pp. 579-589, December 1982.

[Gus85] Gustafson, David. E. "Single Photon Positron Emission-Computed Tomography," in Robb, Richard A. (Ed): *Three-Dimensional Biomedical Imaging*, vol. 2, Boca Raton, Florida: CRC Press, Inc., pp. 1-40, 1985.

[Hae84] Haerten, R.L. and Hernandez, T. "Single Photon Emission Computed Tomography (SPECT): Principles, Technical Implementation, and Clinical Application," *Electromedica*, vol. 52, no. 2, pp. 66-80, 1984.

[Hai86] Haines, Eric A. and Greenberg, Donald P. "The Light Buffer: A Shadow-Testing Accelerator," *IEEE Computer Graphics and Applications*, vol. 6, no. 9, pp. 6-16, September 1986.

[Hal88] Halama, James R.; Henkin, Robert E.; and Friend Leslie E. "Gamma Camera Radionuclide Images: Improved Contrast with Energy-weighted Acquisition," *Radiology*, vol. 169, no. 2, pp. 533-538, November 1988.

[Hara72] Harary, Frank. *Graph Theory*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1972.

[Hard84] Hardas, Dilip M., and Srihari, Sargur N. "Progressive Refinement of 3-D Images Using Coded Binary Trees: Algorithms and Architecture," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 6, pp. 748-756, November 1984.

[Harr79] Harris, L.D.; Robb, R.A.; Yuen, T.S.; and Ritman,E.L. "Display and Visualization of Three-Dimensional Reconstructed Anatomic Morphology: Experience with the Thorax, Heart, and Coronary Vasculature of Dogs," *Journal of Computer Assisted Tomography*, vol. 3, no. 4, pp. 439-446, August 1979.

[Harr86] Harris, L.D.; Camp, J.J.; Ritman, E.L.; and Robb, R.A. "Three-Dimensional Display and Analysis of Tomographic Volume Images Utilizing a Varifocal Mirror," *IEEE Transactions on Medical Imaging*, vol. MI-5, No. 2, pp. 67-72, June 1986.

[Hay87] Hayes, John P.; Mudge, Trevor N.; Stout, Quentin F.; Colley Stephen; and Palmer, John. "Architecture of a Hypercube Supercomputer," *Hypercube Multiprocessors 1987: Proceedings of the International Conference on Parallel Processing*, pp. 653-660, 1986.

[Hay87] Hayes, J.P.; Jain, R.; Martin, W.R.; Mudge, T.N.; Scott, L.R.; Shin, K.G.; and Stout, Q.F. "Hypercube Computer Research at the University of Michigan," *Hypercube Multiprocessors 1987: Proceedings of the Second Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 382-394, September 29-October 1, 1986.

[Hef85a] Heffernan, Patrick B., and Robb, Richard A. "A New Method for Shaded Surface Display of Biological and Medical Images," *IEEE Transactions on Medical Imaging*, vol. MI-4, no. 1, pp. 26-38, March 1985.

[Hef85b] Heffernan, Patrick B. and Robb, Richard A. "Display and Analysis of 4-D Medical Images," *Proceedings of the International Symposium: CAR '85, Computer Assisted Radiology*, Berlin, West Germany, pp. 583-592, 1985.

[Her78] Herman, G.T., and Liu, H.K. "Dynamic Boundary Surface Detection," *Computer Graphics and Image Processing*, vol. 7, pp. 130-138, 1978.

[Her79] Herman, Gabor T., and Liu, Hsun Kao. "Three-Dimensional Display of Human Organs from Computed Tomograms," *Computer Graphics and Image Processing*, vol. 9, pp. 1-21, 1979.

[Her80a] Herman, Gabor T., and Coin, C. Gene. "The Use of Three-Dimensional Computer Display in the Study of Disk Disease," *Journal of Computer Assisted Tomography*, vol. 4, no. 4, pp. 564-567, August 1983.

[Her80b] Herman, Gabor T. *Image Reconstruction From Projections: The Fundamentals of Computerized Tomography,* New York, New York: Academic Press, 1980.

[Her82] Herman, Gabor T; Udupa, Jayaram K.; Kramer, David; Lauterbur, Paul C.; Rudin, Andrew M.; and Schneider, Jay S. "Three-Dimensional Display of Nuclear Magnetic Resonance Images," *Optical Engineering*, vol. 21, no. 5, pp. 923-926, September/October 1982.

[Her83] Herman, Gabor T., and Webster, Dallas. "A Topological Proof of a Surface Tracking Algorithm," *Computer Vision, Graphics, and Image Processing*, vol. 23, pp. 162-177, 1983.

[Her85a] Herman, Gabor, T. "Computer Graphics in Radiology," *Proceedings of the International Symposium: CAR '85, Computer Assisted Radiology*, Berlin, West Germany, pp. 539-550, 1985.

[Her85b] Herman, Gabor T. "X-Ray Computed Tomography - Basic Principles," in Robb, Richard A. (Ed): *Three-Dimensional Biomedical Imaging*, vol. 1, CRC Press, Inc., Boca Raton, Florida, pp. 61-80, 1985.

[Her85c] Herman, Gabor T. “Reconstruction Algorithms,” in Reivich, Martin and Alavi, Abass (Eds): *Positron Emission Tomography*, New York, New York: Alan R. Liss, Inc, pp. 103-117, 1985.

[Her86] Herman, Gabor T. "Computerized Reconstruction and 3-D Imaging in Medicine," *Technical Report MIPG108*, Dept. of Radiology, University of Pennsylvania, February 1986.

[Hic85] Hichwa, R.D. "Positron Emission Tomography: Use of Short-Lived Radionuclides for Neurological Research," *Nuclear Instruments and Methods in Physics Research*, vol. B10-11, no. 2, pp 1072-1076, May 1985.

[Hil85] Hill, Barbara C. and Hinshaw, Waldo S. "Fundamentals of NMR Imaging," in Robb, Richard A. (Ed): *Three-Dimensional Biomedical Imaging*, vol. 2, Boca Raton, Florida: CRC Press, Inc., pp. 79-124, 1985.

[Hil86] Hill, C.R. *Physical Principles of Medical Ultrasonics,* New York, New York: John Wiley & Sons, 1986.

[Hin83] Hinshaw, Waldo S. and Lent, Arnold H. "An Introduction to NMR Imaging: From the Bloch Equation to the Imaging Equation," *Proceedings of the IEEE*, vol. 71, no. 3, pp 338-350, March 1983.

[Hob88] Hobbie, Russell K. *Intermediate Physics for Medicine and Biology*. New York, New York: John Wiley & Sons, 1988.

[Hoh86] Hohne, Karl Heinz, and Bernstein, Ralph. "Shading 3D-Images from CT Using Gray-Level Gradients," *IEEE Transactions on Medical Imaging*, vol. MI-5, no. 1, pp. 45-47, March 1986.

[Hoh89] Hohne, K.H.; Bomans, M.; Pommert, A.; Reimer, M.; Schiers, C.; Tiede, U.; and Wiebecke, G. "3D-Visualization of Tomographic Volume Data Using the Generalized Voxel-Model," *Proceedings of the Chapel Hill Workshop on Volume Visualization*, Chapel Hill, North Carolina, pp. 51-57, May 18-19, 1989.

[Hos86] Hoshino, Tsutomu. "An Invitation to the World of PAX," *Computer*, pp. 68-79, May 1986.

[Hos87] Hoshino, Tsutomu; Shirakawa; Tomonori; and Tsuboi, Keiichi. "Mesh-connected parallel computer PAX for scientific applications," *Parallel Computing*, vol. 5, pp. 363-371, 1987.

[Hu89] Hu, X; Tan, K.K.; Levin, D.N.; Galhotra, S.G.; Pelizzari, C.A.; Chen, G.T.Y.; Beck, R.N.; Chen, C-T; and Cooper, M.D. "Volumetric Rendering of Multimodality, Multivariable Medical Imaging Data," *Proceedings of the Chapel Hill Workshop on Volume Visualization,* Chapel Hill, North Carolina, pp. 45-49, May 18-19, 1989.

[Hun79] Hunter, Gregory M., and Steiglitz, Kenneth. "Operations on Images Using Quad Trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-1, no. 2, pp. 145-153, April 1979.

[Jac80] Jackins, Chris L., and Tanimoto, Steven L. "Oct-trees and Their Use in Representing Three-Dimensional Objects," *Computer Graphics and Image Processing*, vol. 14, pp. 249-270, 1980.

[Jas80] Jaszczak, Ronald, J.; Coleman, R. Edward; and Kim, Chun Bin. "SPECT: Single Photon Emission Computed Tomography," *IEEE Transactions on Nuclear Science*, vol. NS-27, no. 3, pp. 1137-1153, June 1980.

[Jas88] Jaszczak, Ronald, J. "Tomographic Radiopharmaceutical Imaging," *Proceedings of the IEEE*, vol. 76, no. 9, pp. 1079-1094, September, 1988.

[Joh89] Johnson, E. Ruth and Mosher, Charles E. Jr. "Integration of Volume Rendering and Geometric Graphics," *Proceedings of the Chapel Hill Workshop on Volume Visualization*, Chapel Hill, North Carolina, pp.1-7, May 18-19, 1989.

[Kak87] Kak, Avinash C. and Slaney, Malcolm. *Principles of Computerized Tomographic Imaging*. New York, New York: IEEE Press, 1987.

[Kak88] Kak, Avinash C., and Slaney, Malcolm. *Principles of Computerized Tomographic Imaging*. New York, New York: The Institute of Electrical and Electronics Engineers, 1988.

[Kan88] Kanal, Emanuel and Wehrli, Felix W. "Signal-to-Noise Ratio, Resolution, and Contrast" In Wehrli, Felix W.; Shaw, Derek; and Kneeland, Bruce J. (Eds): *Biomedical Magnetic Resonance Imaging*, New York, New York: VCH Publishers, Inc., pp. 47-112, 1988.

[Kap63] Kaplan, Irving. *Nuclear Physics*, 2nd. edition, New York, New York: Addison Wesley Pub. Co., 1963.

[Kap79] Kaplan, Michael, and Greenberg, Donald P. "Parallel Processing Techniques for Hidden Surface Removal," *Computer Graphics*, vol. 13, no. 2, pp. 300-307, August 1979.

[Kap87] Kaplan, Michael R. "The Use of Spatial Coherence in Ray Tracing," in Rogers, David F. and Earnshaw, Rae A. (Eds): *Techniques for Computer Graphics*, New York, New York: Springer-Verlag, pp. 173-190, 1988.

[Kar88] Karp, Joel S. and Muehllehner, Gerd. "Positron Emission Tomography," In Newhouse, Vernon L. (ed): *Progress in Medical Imaging*, New York, New York: Springer Verlag, pp. 185-214, 1988.

[Kau85] Kaufman, Arie and Bakalash, Reuven. "A 3-D Cellular Frame Buffer," in Vandoni, C.E. (ed.): *Proceedings of EUROGRAPHICS '85*, Amsterdam, The Netherlands: Elsevier Science Publishers B.V. (North-Holland), pp. 215-220, 1985.

[Kau86a] Kaufman, Arie. "Voxel-Based Architectures for Three-Dimensional Graphics," *Proceedings of IFIP'86, 10th World Computer Congress*, Dublin, Ireland, pp. 361-366, September 1986.

[Kau86b] Kaufman, Arie and Shimony, Eyal. "3D Scan-Conversion Algorithms for Voxel-Based Graphics," *Proceedings of the ACM Workshop on Interactive 3D Graphics*, Chapel Hill, North Carolina, pp. 45-75, October 23-24, 1986.

[Kau87] Kaufman, Arie. "Efficient Algorithms for 3D Scan-Conversion of Parametric Curves, Surfaces, and Volumes," *IEEE Computer Graphics*, vol. 21, no.4, pp. 171-179, July 1987.

[Kau88a] Kaufman, Arie. "Efficient Algorithms for Scan-Converting 3D Polygons," *Computers and Graphics*, vol. 12, no. 2, pp. 213-219, 1988.

[Kau88b] Kaufman, Arie. "The Cube Workstation - a 3-D voxel-based graphics environment," *The Visual Computer*, vol. 4, no. 4, pp. 210-221, October 1988.

[Lev88] Levoy, Marc. "Display of Surfaces from Volume Data," *IEEE Computer Graphics and Applications*, vol. 8, no. 5, pp. 29-37, May 1988.

[Lev89a] Levoy, Marc. *Display of Surfaces from Volume Data*, PhD. Dissertation, Dept. of Computer Science, University of North Carolina at Chapel Hill, May 1989, 79pp.

[Lev89b] Levoy, Marc. "Design for a Real-Time High-Quality Volume Rendering Workstation," *Proceedings of the Chapel Hill Workshop on Volume Visualization*, Chapel Hill, North Carolina, pp.85-92, May 18-19 1989.

[Lit83] Littleton, Jesse T. and Durizch, Mary L. *Sectional Imaging Methods: A Comparison.* Baltimore, Maryland: University Park Press, 1983.

[Liu77] Liu, Hsun K. "Two- and Three-Dimensional Boundary Detection," *Computer Graphics and Image Processing*, vol. 6, pp. 123-134, 1977.

[Lor87] Lorensen, William E. and Cline, Harvey E. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, vol. 21, no. 4, pp. 163-169, July 1987.

[Lou80] Lougheed, Robert M., and McCubbrey, David L. "The Cytocomputer: A Practical Pipelined Image Processor," *Conference Proceedings of the Seventh Annual Symposium on Computer Architecture*, LaBaule, France, pp. 271-277, May 1980.

[Lun87] Lundstrom, Stephen F. "Applications Considerations in the System Design of Highly Concurrent Multiprocessors," *IEEE Transactions on Computers*, vol. C-36, no. 11, pp. 1292-1308, November 1987.

[Lue80] Luetjen, Karl; Gemmar, Peter; and Ischen, Hartmut. "FLIP: A Flexible Multiprocessor System for Image Processing," *Proceedings of the Fifth International Conference on Pattern Recognition*, Miami Beach, Florida, pp. 326-328, December 1980.

[Mac83] Macovski, Albert. "Physical Problems of Computerized Tomography," *Proceedings of the IEEE*, vol. 71, no. 3, pp 373-378, March 1983.

[Mao87] Mao, Xiaoyang; Kunii, Tosiyasu L.; Fujishiro, Issei; and Noma, Tsukasa. "Hierarchical Representations of 2D/3D Gray-Scale Images and Their 2D/3D Two-Way Conversion," *IEEE Computer Graphics and Applications*, vol. 7, no. 12, pp. 37-44, December 1987.

[McD81] McDicken, W.N. *Diagnostic Ultrasonics: Principles and Use of Instruments.* New York, New York: John Wiley & Sons, 1981.

[McL88] McLeod, Jonah. "Ardent Launches `First Supercomputer on a Desk'," *Electronics*, pp. 65-67, March 3, 1988.

[Mea82a] Meagher, Donald. "Efficient Synthetic Image Generation of Arbitrary 3D Objects," *Proceedings of the 1982 IEEE Computer Society Conference on Pattern Recognition and Image Processing*, Las Vegas, Nevada, pp. 473-478, June 14-17, 1982.

[Mea82b] Meagher, Donald. "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing*, vol. 19, pp. 129-147, 1982.

[Mey87] Meyers, Robert A., ed. *Encyclopedia of Physical Science and Technology*. Orlando, Florida: Academic Press, 1987. s.v. "Computerized Tomography," by Z. H. Cho and "Ultrasonics," by John Szilard.

[Mil85] Miller, Russ, and Stout, Quentin F. "Geometric Algorithms for Digitized Pictures on a Mesh-Connected Computer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-7, no. 2, pp. 216-228, March 1985.

[Mil87] Miller, Russ, and Stout, Quentin F. "Some Graph- and Image-Processing Algorithms for the Hypercube," *Hypercube Multiprocessors 1987: Proceedings of the Second Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 418-425, September 29-October 1, 1986.

[Mil89] Mills, Peter H.; Fuchs, Henry; Pizer, Stephen; and Rosenman, Julian G. "IMEX: a tool for image display and contour management in a windowing environment," To appear in: *Proceedings of the SPIE Conference on Medical Imaging III*, Newport Beach, California, January 29-February 3, vol. 1091, 1989.

[Mir88] Miranker, Glen S.; Rubinstein, Jon; and Sanguinetti, John. "Squeezing a Cray-Class Supercomputer Into a Single-User Package," *Thirty-Third IEEE Computer Society International Conference*, San Francisco, California, pp. 452-456, February 29 - March 4, 1988.

[Mir84] Mirell, Stuart G. "Computed Tomography: Relations to Nuclear Medicine," In Harbert, John and DaRocha, Antonio F. G., (Eds): *Textbook of Nuclear Medicine, Volume I: Basic Science*, 2nd edition, Philadelphia, Pennsylvania: Lea & Febiger, pp 408-423, 1984.

[Mol88] Moler, Cleve. "Single-User Supercomputers or How I Got Rid of the BLAS," *Thirty-Third IEEE Computer Society International Conference*, San Francisco, California, pp. 448-451, February 29 - March 4, 1988.

[Mon87] Montani, Claudio and Re, Michele. “Vector and Raster Hidden-Surface Removal Using Parallel Connected Stripes,” *IEEE Computer Graphics and Applications*, vol. 7, no. 7, pp. 48-68, July 1987.

[Mon86] Montry, G.R. "A Portable System for Simulating Distributed Parallel Processors on Shared Memory Machines," *Hypercube Multiprocessors 1987: Proceedings of the Second Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 276-282, September 29-October 1, 1986.

[Mor81] Morgenthaler, David G., and Rosenfeld, Azriel. "Multidimensional Edge Detection by Hypersurface Fitting," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-3, no.4, pp. 482-486, July 1981.

[Mor86] Morris, Peter G. *Nuclear Magnetic Resonance Imaging in Medicine and Biology*, Oxford, England: Clarendon Press, 1986.

[Mu87] Mu, Zhuing and Chen, Marina C. "Communication-Efficient Distributed Data Structures on Hypercube Machines," *Hypercube Multiprocessors 1987: Proceedings of the Second Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 67-77, September 29-October 1, 1986.

[Mud87] Mudge, T.N., and Abdel-Rahman, T.S. "Vision Algorithms for Hypercube Machines," *Journal of Parallel and Distributed Computing*, vol. 4, no. 1, pp. 79-94, 1987.

[Mue86] Muehllehner, Gerd and Karp, Joel S. "A Positron Camera Using Position-Sensitive Detectors: PENN-PET," *The Journal of Nuclear Medicine*, vol. 27, no. 1, pp. 90-98, January, 1986.

[Mue88] Muehllehner, Gerd and Karp, Joel S. "Advances in PET and SPECT," *IEEE Transactions on Nuclear Science*, vol. 35, no. 1, pp. 639-643, February, 1988.

[Mur80] Murphy, Paul H.; DePuey, E. Gordon; Sonnemaker, Robert E.; and Burdine, John A. "Emission Computed Tomography: A Current Status Report," In Freeman, Leonard M. and Weissman, Heidi S., (Eds): *Nuclear Medicine Annual 1980*, New York, New York: Raven Press, pp. 83-125, 1980.

[Nat86] Natterer, F. *The Mathematics of Computerized Tomography,* Chichester, England: John Wiley & Sons, 1986.

[Ncu85] NCUBE Corporation. *NCUBE/ten: An Overview*, November 1985.

[Ncu89] NCUBE Corporation. *NCUBE Supercomputers*, June 1989.

[New83] Newton, Thomas H. and Potts, D. Gordon, eds. *Modern Neuroradiology*. Vol. 2, *Advanced Imaging Techniques*. San Anselmo, California: Clavadel Press, 1983.

[Nic86] Nickel, O. and Hahn, K. "New Nuclear Medical Technologies for Pediatrics," *Medical Progress Through Technology*, vol. 2, no. 2, pp. 65-72, 1986.

[Old88] Oldendorf, William H., and Oldendorf, William H. Jr. *Basics of Magnetic Resonance Imaging,* Boston, Massachusetts: Martinus Nijhoff Publishing, 1988.

[Orp88] Orphanoudakis, Stelios C. "Supercomputing in Medical Imaging," *IEEE Engineering in Medicine and Biology Magazine*, vol. 7, no. 10, pp. 16-20, December 1988.

[Osw85] Oswald, H. "A Medical Workstation for the Three-Dimensional Display of Computed Tomogram Images," *Proceedings of the International Symposium: CAR '85, Computer Assisted Radiology*, Berlin, West Germany, pp. 5565-571, 1985.

[Paa85] Paans, A.M.J.; Vaalburg, W.; and Woldring, M.G. "A Rotating Double-Headed Positron Camera," *The Journal of Nuclear Medicine*, vol. 26, no. 12, pp. 1466-1471, December, 1985.

[Par80] Parke, Frederic I. "Simulation and Expected Performance Analysis of Multiple Processor Z-Buffer Systems," *Computer Graphics*, vol. 14, no. 3, pp. 175-181, July 1980.

[Pat89a] Patton, James A. "Nuclear Medicine, Physics and Instrumentation," In Sandler, Martin P.; Patton, James A.; Shaff, Max I.; Powers, Thomas A.; and Partain, C. Leon. *Correlative Imaging: Nuclear Medicine, Magnetic Resonance, Computed Tomography, Ultrasound*, Baltimore, Maryland: Williams and Wilkins, pp. 3-30, 1989.

[Pat89b] Patton, James A. "Physics and Instrumentation of Magnetic Resonance Imaging," In Sandler, Martin P.; Patton, James A.; Shaff, Max I.; Powers, Thomas A.; and Partain, C. Leon. *Correlative Imaging: Nuclear Medicine, Magnetic Resonance, Computed Tomography, Ultrasound*, Baltimore, Maryland: Williams and Wilkins, pp. 31-48, 1989.

[Pet85] Peterson, J.C.; Tuazon, J.O.; Lieberman, D.; and Pniel, M. "The Mark III Hypercube Concurrent Processor," *Proceedings of the International Conference on Parallel Processing*, pp. 71-73, 1985.

[Phe75] Phelps, M.E.; Hoffman, E.J.; Mullani, N.A.; and Ter-Pogossian, M.M. "Application of Annihilation Coincidence Detection to Transaxial Reconstruction Tomography," *Journal of Nuclear Medicine*, vol. 16, no. 3, March 1975, pp. 210-224.

[Phe85a] Phelps, Michael E. and Mazziotta, John C. "Positron Emission Tomography: Human Brain Function and Biochemistry," *Science*, vol. 228, no. 4701, pp. 799-809, 17 May 1985.

[Phe85b] Phelps, Michael E.; Mazziotta, John C.; Schelbert, Heinrich R.; Hawkins, Randall A.; and Engel, Jerome Jr. "Clinical PET: What are the Issues?," *The Journal of Nuclear Medicine*, vol. 26, no. 12, pp. 1353-1358, December 1985.

[Pho75] Phong, Bui Tuong. "Illumination for Computer Generated Graphics," *Communications of the ACM*, vol. 18, no. 6, pp. 311-317, June 1975.

[Pic89] Pickens, David R. III and Price, Ronald R. "Physics and Instrumentation of Correlative Modalities, In Sandler, Martin P.; Patton, James A.; Shaff, Max I.; Powers, Thomas A.; and Partain, C. Leon. *Correlative Imaging: Nuclear Medicine, Magnetic Resonance, Computed Tomography, Ultrasound*, Baltimore, Maryland: Williams and Wilkins, pp. 49-62, 1989.

[Pix88a] PIXAR™ Corporation. *ChapLibraries™ Technical Summary for the UNIX Operating System*, December 1988.

[Pix88b] PIXAR™ Corporation. *ChapTools™ Technical Summary for the UNIX Operating System*. December 1988.

[Pix88c] PIXAR™ Corporation. *Pixar II™ Hardware Overview*. February 1988.

[Pix88d] PIXAR™ Corporation. *Chap™ Image Processing System Technical Summary*. November 1988.

[Pix88e] PIXAR™ Corporation. *ChapVolumes™ Volume Rendering Package Technical Summary*. January 1989.

[Piz84] Pizer, Stephen M.; Zimmerman, John B.; and Staab, Edward V. "Adaptive Grey Level Assignment in CT Scan Display," *Journal of Computer Assisted Tomography*, vol. 8, no. 2, pp. 300-305, April 1984.

[Piz85] Pizer, Stephen M. and Fuchs, Henry. "Three Dimensional Image Presentation Techniques in Medical Imaging," *Proceedings of the International Symposium: CAR '87, Computer Assisted Radiology*, Berlin, West Germany, pp. 589-598, 1987.

[Piz87] Pizer, Stephen M.; Amburn, E. Philip; Austin, John D.; Cromartie, Robert; Geselowitz, Ari; Greer, Trey; Romeny, Bart ter Haar; Zimmerman, John B.; and Zuiderveld, Karel. "Adaptive Histogram Equalization and Its Variations," *Computer Vision, Graphics, and Image Processing*, vol. 39, no. 3, pp. 355-368, September 1987.

[Pod87] Podo, F. "Technological and Clinical Evaluation of MRI and MRS," *IEEE Ninth Annual Conference of the Engineering in Medicine and Biology Society*, vol. 4, no. 1, pp. 21-25, 1987.

[Por84] Porter, Thomas and Duff, Tom. "Compositing Digital Images," *Computer Graphics*, vol. 18, no. 3, pp. 253-259, July 1984.

[Pot81] Potter, J.L. "Continuous Image Processing on the MPP," *1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Hot Springs, Virginia, pp. 51-56, November 1981.

[Pou87] Poulton, John; Fuchs, Henry; Austin, John; Eyles,John; and Greer, Trey. "Building a 512x512 Pixel-Planes System," *1987 Conference on Advanced Research in VLSI*, pre-print, Stanford University, Palo Alto, CA, March 1987.

[Pre82] Preston, Kendall Jr., and Uhr, Leonard, (eds.), *Multicomputers and Image Processing: Algorithms and Programs*. New York, New York: Academic Press, 1982.

[Pyk82] Pykett, Ian L; Newhouse, Jeffrey H.; Buonanno; Ferdinando S.; Brady, Thomas J.; Goldman, Mark R.; Kistler, J. Philip; and Pohost, Gerald M. "Principles of Nuclear Magnetic Resonance Imaging," *Radiology*, vol. 143, no. 1, pp. 157-168, April 1982.

[Ran88] Ranka, Sanjay; Won, Youngju; and Sahni, Sartag. "Programming the NCUBE Hypercube," *Technical Report 88-13*, Computer Science Department, 136 Lind Hall, University of Minnesota, January 1988. To appear in *IEEE Software*.

[Red81] Redington, Rowland W. and Berninger, Walter H. "Medical Imaging Systems," *Physics Today*, vol. 34, no. 8, pp. 36-44, August 1981.

[Ree84] Reeves, Anthony P. "Parallel Computer Architectures for Image Processing," *Computer Vision, Graphics, and Image Processing*, vol. 25, pp. 68-88, 1984.

[Rey83a] Reynolds, R.A. "Simulation of 3D Display Systems," unpublished report, Dept. of Radiology, University of Pennsylvania, 20 May 1983.

[Rey83b] Reynolds, R.A. "Some Architectures for Real-Time Display of Three-Dimensional Objects: A Comparative Survey," *Technical Report MIPG84*, Dept. of Radiology, University of Pennsylvania, October 1983.

[Rey85] Reynolds, Richard A. *Fast Methods for 3D Display of Medical Objects*, Ph.D. Dissertation, Dept. of Computer and Information Science, University of Pennsylvania, May 1985, 257pp.

[Rey87] Reynolds, R. Anthony; Gordon, Dan; and Chen, Lih-Shyang. "A Dynamic Screen Technique for Shaded Graphics Display of Slice-Represented Objects," *Computer Vision, Graphics, and Image Processing*, vol. 38, no. 3, pp. 275-298, June 1987.

[Rey88a] Reynolds, R. A.; Sontag, M.R.; and Chen, LS. "An algorithm for three-dimensional visualization of radiation therapy beams," *Medical Physics*, vol. 15, no. 1, pp. 24- 28, January-February 1988.

[Rey88b] Reynolds, R.A.; Decuypere, P.; Goldwasser, S.M.; Talton, D.A.; Walsh, E.S.; Holshouser, B.A.; and Christiansen, E.L. "Realistic Presentation of Three-Dimensional Medical Datasets," *Proceedings Graphics Interface 1988*, Edmonton, Canada, pp. 71-77, June 1988.

[Rho79] Rhodes, Michael L. "An Algorithmic Approach to Controlling Search in Three-Dimensional Image Data," *Computer Graphics*, vol. 13, no. 2, pp. 134-141, August 1979.

[Rie81] Rieger, Chuck. "ZMOB: Doing it in Parallel," *1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Hot Springs, Virginia, pp. 119-124, November 1981.

[Rob73] Robertson, J.S.; Marr, R.B; Rosenblum, B.; *et. al.*, "Thirty-two Crystal Positron Transverse Section Detector," in *Tomographic Imaging in Nuclear Medicine*, G.S. Freeman, ed. New York, New York: Society of Nuclear Medicine, 1973, pp. 141-153.

[Rob82] Robb, Richard A. "X-Ray Computed Tomography: An Engineering Synthesis of Multiscientific Disciplines," *CRC Critical Reviews in Biomedical Engineering*, vol. 7, no. 4, pp. 265-333, March 1982.

[Rob85a] Robb, Richard A. *Three-Dimensional Biomedical Imaging*. vol. 1. Boca Raton, Florida: CRC Press, 1985.

[Rob85b] Robinson, Brent S. and Greenleaf, James F. "Computerized Ultrasound Tomography," in Robb, Richard A. (Ed): *Three-Dimensional Biomedical Imaging*, vol. 2, Boca Raton, Florida: CRC Press, Inc., pp. 57-78, 1985.

[Rob86a] Robb, R.A.; Hefferman, P.B.; Camp, J.J.; and Hanson,D.P. "A Workstation for Interactive Display and Quantitative Analysis of 3D and 4D Biomedical Images," *Proceedings of the Tenth Annual Symposium on Computer Applications in Medical Care*, Washington, D.C., pp. 240-256, October 25-26, 1986.

[Rob86b] Robertson, Barbara. "Pixar Goes Commercial in a New Market," *Computer Graphics World*, vol. 9, no. 6, pp. 61-70, June 1986.

[Rob87] Robb, Richard A. "A Workstation for Interactive Display and Analysis of Multidimensional Biomedical Images," *Proceedings of the International Symposium: CAR '87, Computer Assisted Radiology*, Berlin, West Germany, pp. 642-656, 1987.

[Rob88] Robb, Richard A. and Barillot, C. "Interactive 3-D Image Display and Analysis," in Casasent, D.P. and Tescher, A.G. (Eds.): *Proceedings of the Society of Photo-Optical Instrumentation Engineer: Hybrid Image and Signal Processing*, vol. 939, Orlando, Florida, pp. 173-202, 1988.

[Rog85] Rogers, David F. *Procedural Elements for Computer Graphics.* New York, New York: McGraw-Hill, 1985.

[Rol82] Rollo, F. David. "New Imaging Modalities: Positron Emission Tomography and Nuclear Magnetic Resonance," *Medical Instrumentation*, vol. 16, no. 1, pp. 53-54, January-February 1982.

[Ros82] Rosenfeld, Azriel, and Kak, Avinash C. *Digital Picture Processing.* Orlando, Florida: Academic Press, 1982.

[Ros89] Rossignac, Jaroslaw R. "Considerations on the Interactive Rendering of Four-Dimensional Volumes," *Proceedings of the Chapel Hill Workshop on Volume Visualization*, Chapel Hill, North Carolina, pp. 67-76, May 18-19, 1989.

[Run84] Runge, Val M.; Partain, C. Leon; and James, A. Everette Jr. "Magnetic Resonance-Relations to Nuclear Medicine," In Harbert, John and DaRocha, Antonio F. G., (Eds): *Textbook of Nuclear Medicine, Volume I: Basic Science*, 2nd edition, Philadelphia, Pennsylvania: Lea & Febiger, pp 424-437, 1984.

[Sad87] Sadayappan, Ponnuswamy, and Ercal, Fikret. "Nearest-Neighbor Mapping of Finite Element Graphs onto Processor Meshes," *IEEE Transactions on Computers*, vol. C-36, no. 12, pp. 1408-1424, 1987.

[Sam86] Samet, Hanan, and Tamminen, Markku. "An Improved Approach to Connected Component Labeling of Images," *Proceedings of the 1986 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Miami Beach, Florida, pp. 312-318, June 22-26, 1986.

[Sam88] Samet, Hanan and Webber, Robert E. "Hierarchical Data Structures and Algorithms for Computer Graphics," *IEEE Computer Graphics and Applications*, vol.8, no. 3, pp. 48-68, May 1988.

[Sca89] Scanditronix, *Scanditronix Positron Emission Tomography Systems*, 1989.

[Sch86] Scheifler, Robert W. and Gettys, Jim. "The X Window System," *ACM Transactions on Graphics*, vol. 5, no. 2, pp. 79-109, April 1986.

[Sch87] von Schulthess, G.K. and Jung, T. "MRI: Recent Advances and Their Potential Impact on Motion Artifact Reduction," *Medicamundi*, vol. 32, no. 2, pp. 49-54, 1987.

[Sch87] Schwan, Karsten; Bo, Win; Bauman, N.; Sadayappan, P.; and Ercal, F. "Mapping Parallel Applications to a Hypercube," *Hypercube Multiprocessors 1987: Proceedings of the Second Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 141-151, September 29-October 1, 1986.

[Sco87] Scott, L.R.; Boyle, J.M.; and Bagheri, B. "Distributed Data Structures for Scientific Computation," *Hypercube Multiprocessors 1987: Proceedings of the Second Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 55-66, September 29 - October 1, 1986.

[Sei83] Seitz, Peter, and Ruegsegger, Peter. "Fast Contour Detection Algorithm for High Precision Quantitative CT," *IEEE Transactions on Medical Imaging*, vol. MI-2, no. 3, pp. 136-141, September 1983.

[Sei85] Seitz, C.L. "The Cosmic Cube," *Communications of the ACM*, vol. 28, no. 1, pp. 22-33, January 1985.

[Sha88] Shabestari, Behrouz N. and Farison, James B. "Iterative Methods of Tomographic Image Reconstruction in Medical Imaging," *Proceedings of the Tenth Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, New Orleans, Louisiana, pp. 416-417, November 4-7, 1988.

[Sor87] Sorenson, James A. and Phelps, Michael E. *Physics in Nuclear Medicine*. Orlando, Florida: Grune & Stratton, Inc., 1987.

[Sou83] Soussaline, F. and Comar, D. "Positron Emission Tomography: Applications to the Study of Metabolic and Pathophysiologic Mechanisms," *Nuclear Science Applications*, vol. 1, pp. 601-616, 1983.

[Spo88] Sporer, Michael; Moss, Franklin H.; and Mathias, Craig J. "An Introduction to the Architecture of the Stellar™ Graphics Supercomputer," *Proceedings of the Thirty-Third IEEE Computer Society International Conference*, San Francisco, California, pp. 464-467, February 29 - March 4, 1988.

[Spr86] Springer, David. PIXAR™ Image Computer Revolutionizes Industrial Design," *Intelligent Instruments & Computers*, pp. 276-277, November/December 1986.

[Spr87] Sprawls, Perry Jr. *Physical Principles of Medical Imaging*, Rockville, Maryland, Aspen Publishers, Inc, 1987.

[Sri81] Srihari, Sargur N. "Representation of Three-Dimensional Digital Images," *Computing Surveys*, vol. 13, no. 4, pp. 399-424, December 1981.

[Sri84] Srihari, Sargur N. "Pyramid Representation for Solids," *Information Sciences*, vol. 34, pp.25-46, 1984.

[Sta88] Stark, David D. and Bradley, William G. *Magnetic Resonance Imaging*, St. Louis, Missouri: The C. V. Mosby Company, 1988.

[Ste79] Sternberg, Stanley R. "Parallel Architectures for Image Processing," *COMPSAC '87, The IEEE Computer Society's Third International Computer Software and Applications Conference*, Chicago, Illinois, pp. 712-717, November 1979.

[Ste88] Stellar Computer Inc. *Stellar Graphics Supercomputer Model GS1000™ System Overview*. 1988.

[Ste89a] Stellar Computer Inc. *GS2000™ Series Backgrounder*. 1989.

[Ste89b] Stellar Computer Inc. *Stellar Computer Inc. Product Family*. 1989.

[Sto83] Stout, Quentin F. "Sorting, Merging, Selecting, and Filtering on Tree and Pyramid Machines," *Proceedings of the 1983 International Conference on Parallel Processing*, Columbus, Ohio, pp. 214-221, August 23-26, 1983.

[Sto87] Stout, Quentin F., and Wagar, Bruce. "Passing Messages in Link-Bound Hypercubes," *Hypercube Multiprocessors 1987: Proceedings of the Second Conference on Hypercube Multiprocessors*, Knoxville, Tennessee, pp. 251-257, September 29-October 1, 1986.

[Sty88] Stytz, Martin R., Frieder, G., and Frieder, O. "On the Exploitation of A Commercially Available Parallel Processing Architecture for Medical Imaging," *Proceedings of the Symposium on the Engineering of Computer-Based Medical Systems*, Minneapolis, Minnesota, pp. 49-59, June 8-10, 1988.

[Sun88a] Sun Microsystems, Inc. *TAAC-1™ Application Accelerator Technical Notes*, 1988.

[Sun88b] Sun Microsystems, Inc. *Sun's Software Overview*, 1988.

[Sun88c] Sun Microsystems, Inc. *Sun's TAAC-1™ Application Accelerator*, 1988.

[Sun88d] Sun Microsystems, Inc. *TAAC-1™ Application Accelerator: Release 2.2, Revision A*, 15 September, 1988.

[Sun88e] Sun Microsystems, Inc. *TAAC-1™ Performance Benchmarks, Tech Note #16*, October 18, 1988.

[Swa86] Swanson, Roger W. and Thayer, Larry J. "A Fast Shaded-Polygon Renderer," *Computer Graphics*, vol. 20, no. 4, pp. 95-101, August 1986.

[Tal87] Talton, D.A; Goldwasser, S.M.; Reynolds, R.A.; and Walsh, E.S. "Volume Rendering Algorithms for the Presentation of 3-D Medical Data," *NCGA '87 Technical Session Proceedings*, pp. 119-128, 1987.

[Tan83] Tanimoto, Steven L. "A Pyramidal Approach to Parallel Processing," *Conference Proceedings of the Tenth Annual International Symposium on Computer Architecture*, Stockholm, Sweden, pp. 372-378,1983.

[Tay85] Taylor, Kenneth J.W. *Atlas of Ultrasonography,* 2nd Ed., New York, New York: Churchill Livingstone, 1985.

[Ter75] Ter-Pogossian, M.M.; Phelps, M.E.; Hoffman, E.J.; and Mullani, N.A. "A Positron Emission Transaxial Tomograph for Nuclear Medicine Imaging (PETT)," *Radiology*, vol. 114, no. 1, January 1975, pp. 89-98.

[Ter85] Ter-Pogossian, Michael M. "Positron Emission Tomography," in Robb, Richard A. (Ed): *Three-Dimensional Biomedical Imaging*, vol. 2, Boca Raton, Florida: CRC Press, Inc., pp. 41-56, 1985.

[Tie87] Tiede, U.; Hohne, K.H.; and Riemer, M. "Comparison of Surface Rendering Techniques for 3-D Tomographic Objects," *Proceedings of the International Symposium: CAR '87, Computer Assisted Radiology*, Berlin, West Germany, pp. 5610-614, 1987.

[Til88] Till, Johna. "Single-user Supercomputer is Interactive Graphics Powerhouse," *Electronic Design*, vol. 36, no. 8, pp. 57-61, March 1988

[Tod80] Todhunter, J.S. and Li, C.C. "A New Model for Parallel Processing of Serial Images," *Proceedings of the Fifth International Conference on Pattern Recognition*, Miami Beach, Florida, pp. 493-495, December 1980.

[Tod83] Todd-Pokropek, A. "The Mathematics and Physics of Emission Computerized Tomography (ECT)," In Esser, Peter D., (Ed): *Emission Computerized Tomography: Current Trends,* The Society of Nuclear Medicine, New York, New York, pp 3 - 31, 1983.

[Tos88] Toshiba Corporation, *Digital Gamma Camera GCA-901A*, 1988.

[Tos89] Toshiba Corporation, *personal communication with author*, August 1989.

[Tri85a] Trivedi, Sushma S. "Representation of Three-Dimensional Binary Scenes," *Proceedings of the Sixth Annual Conference and Exposition of the National Computer Graphics Association*, Dallas, Texas, pp. 132-144, April 14-18, 1985.

[Tri85b] Trivedi, Sushma S.; Herman, Gabor T.; and Udupa, Jayaram K. "Segmentation into Three Classes Using Gradients," *Technical Report MIPG104, Dept. of Radiology*, University of Pennsylvania, August 1985.

[Tri86] Trivedi, Sushma S.; Herman, Gabor T.; and Udupa Jayaram K. "Segmentation into Three Classes Using Gradients," *IEEE Transactions on Medical Imaging*, vol. MI-5, no. 2, pp. 116-119, June 1986.

[Tsa81] Tsai, W.H.; Chou C.C.; Cheng, P.C.; and Chen, Z. "Architecture of a Multi-Microprocessor System for Parallel Processing of Image Sequences," *1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Hot Springs, Virginia, pp. 104-111, November 1981.

[Tua85] Tuazon, J.; Peterson, J.; Pniel, M.; and Liberman, D. "Caltech Mark II Hypercube Concurrent Processor," *Proceedings of the International Conference on Parallel Processing*, pp. 666-671, 1985.

[Tuo83] Tuomenoksa, David Lee; Adams, George B. III; Siegel,Howard Jay; and Mitchell, O. Robert. "A Parallel Algorithm for Contour Extraction; Advantages and Architectural Implications," *Proceedings of the 1983 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Washington, D.C., pp. 336-344, June 18-23, 1983.

[Tuy84] Tuy, Heang K. and Tuy, Lee Tan. "Direct 2-d Display of 3-d Objects," *IEEE Computer Graphics and Applications*, vol. 4, no. 10, pp. 29-33, October 1984.

[Udu81] Udupa, Jayaram K. "Determination of 3-D Shape Parameters from Boundary Information," *Computer Graphics and Image Processing*, vol. 17, pp. 52-59, 1981.

[Udu82a] Udupa, Jayaram K.; Srihari, Sargur N.; and Herman, Gabor T. "Boundary Detection in Multidimensions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, no. 1, January 1982.

[Udu82b] Udupa, Jayaram K. "Interactive Segmentation and Boundary Surface Formation for 3-D Digital Images," *Computer Graphics and Image Processing*, vol. 18, pp. 213-235, 1982.

[Udu83] Udupa, Jayaram K. "Display of 3D Information in Discrete 3D Scenes Produced by Computerized Tomography," *Proceedings of the IEEE*, vol. 71, no. 3, pp. 420-431, March 1983.

[Uhr82] Uhr, Leonard; Schmitt, Larry; and Hanrahan, Pat. "Cone/Pyramid Perception Programs for Arrays and Networks," draft submitted to *Multicomputers and Image Processing* for the April, 1982 issue.

[Ups88] Upson, Craig and Keeler, Michael. "V-BUFFER: Visible Volume Rendering," *Computer Graphics*, vol. 22, no. 4, pp. 59-64, August 1988.

[Vac88] Vacher, J.; Allemand, R.; Campagnolo, R.; and Lecomte, J.L. "Time of Flight Positron Emission Tomography Characteristics and Technological Parameters, *Nuclear Science Applications*, vol. 3, pp. 97-109, 1988.

[Wei80] Weiman, Carl F.R. "Continuous Anti-Aliased Rotation and Zoom of Raster Images," *Computer Graphics*, vol. 14, no. 3, pp. 286-293, July 1980.

[Wel81] Wells, P.N.T. "A Review of Medical Imaging Methods," *Australasian Physical and Engineering Sciences in Medicine*, vol. 4, no. 2, pp. 33-39, 1981.

[Whi80] Whitted, Turner. "An Improved Illumination Model for Shaded Display," *Communications of the Association for Computing Machinery*, vol. 23, no. 6, pp. 343-349, June 1980.

[Whi85] Whitted, Turner and Cook, Robert L. “A Comprehensive Shading Model,” (unpublished course notes, Tutorial on the State of the Art in Image Synthesis, SIGGRAPH ‘85, course notes #11, San Francisco, California, July 22-26, 1985, 4 pages).

[Wil88] Williams, Tom. "Graphics Supercomputers Tackle Real-time Visualization," *Computer Design*, vol. 27, no. 6, pp. 21-24, March 15 1988.

[Win88] Windham, Joe P.; Abd-Allah, Mahmoud A.; Reimann, David A.; Froelich, Jerry W.; and Haggar, Allan M. "Eigenimage Filtering in MR Imaging," *Journal of Computed Assisted Tomography*, vol. 12, no. 1, pp 1-9, January-February 1988.

[Win84] Winston, Martin A. "Ultrasonography-Relations to Nuclear Medicine," In Harbert, John and DaRocha, Antonio F. G., (Eds): *Textbook of Nuclear Medicine, Volume I: Basic Science*, 2nd edition, Philadelphia, Pennsylvania: Lea & Febiger, pp 390-407, 1984.

[Wix89] Wixson, Steve E. “Volume Visualization of Medical Data,” *Proceedings of the Chapel Hill Workshop on Volume Visualization*, Chapel Hill, North Carolina, pp. 77-83, May 18-19, 1989.

[Won87] Wong, Wilson S.; Tsuruda, Jay S.; Kortman, Keith E.; and Bradley, William G. *Practical MRI: A Case Study Approach*, Rockville, Maryland, Aspen Publishers, Inc, 1987.

[Woo87] Woods, R.E. "Transform-based Processing: How Much Precision is Needed?" *ESD: THE Electronic System Design Magazine*, pp 76-80, February 1987.

[Wor88] Worley, William S. III and Worley, William S. Jr. "Ardent's Fast Memory Meets the Challenge of the Titan Graphics Supercomputer," *VLSI Systems Design*, pp. 50-59, August 1988.

[Wre51] Wren, E.R.; Good, M.L.; and Handler, P. "The Use of Positron Emitting Radioisotopes for the Localization of Brain Tumors," *Science*, vol. 113, no. 2939, April 1951, pp. 525-527.

[Yam85] Yamamoto, Y. Lucas; Thompson, Christopher J.; Diksic, Mirko; and Meyer, Ernest. "Positron Emission Tomography," *Radiation Physics and Chemistry*, vol. 24, no. 3-4, pp. 385-403, 1984.

[Yau83] Yau, Mann-may J. "Hierarchical Representation of Three-Dimensional Digital Objects," *Technical Report No. 201*, Dept. of Computer Science, State University of New York at Buffalo, April 1983.

[Yuv87] Yuval, Gideon. *A voxel-based image segmentation algorithm.* Personal communication with Dr. Gideon Frieder. Relayed to author June, 1987.